relevé des compteurs d'eau basé sur l'IA (1)

intégrez votre ancien compteur dans l'IdO !

Source : Adobe Stock

Daniel Scaini (Italie)

Les compteurs d'eau intelligents sont déjà sur le marché depuis un certain temps, mais pour des raisons techniques ou administratives, le remplacement de nos anciens compteurs n'est généralement pas si simple. Ce projet permet de transformer n'importe quel compteur analogique en compteur numérique à l'aide d'un module ESP32-CAM et d'un système d'intelligence artificielle (IA). Parce que nous allons également dévoiler de nombreux éléments de référence dans cet article, nous l'avons divisé en deux parties.

Elettronica In
WWW.ELETTRONICAIN.IT

L'Italie est l'un des pays européens qui prélève et consomme le plus d'eau à des fins civiles, juste derrière la Grèce. Ces chiffres sont inquiétants, surtout si l'on considère les pénuries d'eau qui frappent ponctuellement notre pays. En effet, la disponibilité moyenne a diminué d'environ 19 % au cours des trois dernières années, aggravée par des températures de plus en plus chaudes et un manque de précipitations. Ce problème n'est pas propre à l'Italie, et plusieurs pays prennent des mesures drastiques, comme la limitation de la consommation, ou le déblocage de financements pour rénover les installations de distribution privées ou publiques afin d'éviter les fuites.

De plus, compte tenu de l'évolution constante de la technologie, plusieurs fournisseurs ont pris des mesures depuis quelques années pour rendre la lecture des compteurs plus efficace chez les particuliers. Cette mesure permet d'éviter le gaspillage de la matière première, mais aussi de réduire les opérations de lecture effectuées maison par maison par les opérateurs.

Dans cet article, nous allons aborder cette thématique, qui consiste à faire un relevé d'un compteur analogique et le numériser, pour qu'il nous transmette les valeurs ou les transmette à un de nos serveurs. Le tout est basé sur une ESP32-CAM, une plateforme déjà connue de beaucoup, et un système d'IA qui peut détecter des photos et les convertir en valeurs probables.

Architecture

Les systèmes d'IA sont entrés massivement dans notre vie quotidienne, il suffit de penser aux assistants vocaux ou à la reconnaissance d'images. Pour les calculs complexes qu'un système d'IA doit effectuer, il est possible de s'appuyer sur l'informatique en nuage (*cloud computing*) sur des plateformes en ligne dédiées, ou de les effectuer directement sur la puce, dans ce que l'on appelle l'informatique en périphérie (edge computing). Avec le perfectionnement croissant des processeurs, ce deuxième mode de traitement se développe et constitue la base de notre projet.

Dans celui-ci, un réseau d'IA et une ESP32-CAM coopéreront afin de pouvoir donner à l'utilisateur un résultat numérique, obtenu en photographiant numériquement un compteur d'eau analogique classique. La reconnaissance et la numérisation sont effectuées par l'ESP32-CAM à l'aide d'un réseau de neurones à convolution (CNN), dont nous parlerons plus loin dans l'article.

La première étape de la mise en œuvre de notre projet consiste à installer le micrologiciel sur notre ESP32-CAM. Ensuite, nous aurons besoin d'une phase de calibration, au cours de laquelle nous identifierons les zones dédiées à la reconnaissance des chiffres et des indicateurs de notre compteur. Une fois cette étape terminée, nous disposerons de toutes les informations nécessaires pour les envoyer numériquement là où nous le souhaitons.

Réseaux de neurones

Les réseaux de neurones sont des modèles informatiques inspirés du fonctionnement du cerveau humain. Ce sont des systèmes d'intelligence artificielle qui tentent d'émuler la façon dont le cerveau traite l'information. Les réseaux neuronaux sont utilisés dans l'apprentissage automatique, un domaine d'étude qui vise à apprendre aux ordinateurs à effectuer certaines tâches sans avoir été expressément programmés à cette fin.

Les réseaux neuronaux jouent un rôle essentiel dans les modèles d'apprentissage profond, une branche de l'apprentissage automatique qui se concentre sur le traitement de

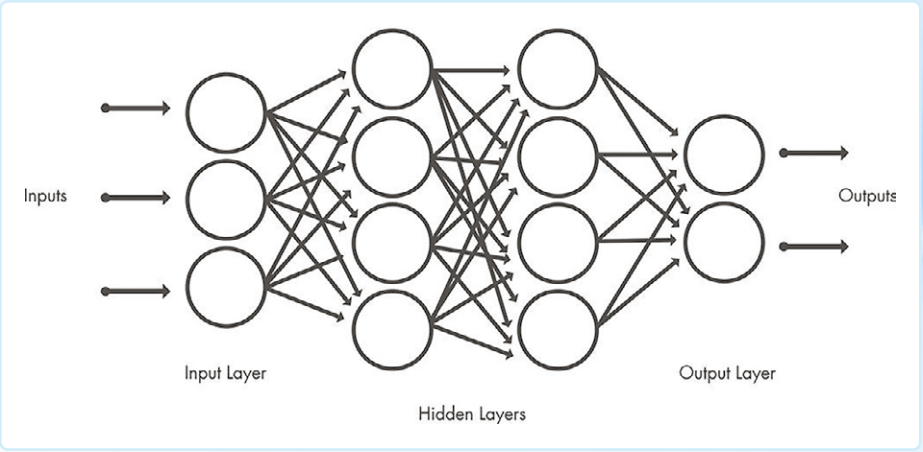

Figure 1. Les réseaux neuronaux sont composés d'une couche d'entrée, d'une couche de sortie et de nombreuses couches intermédiaires appelées couches cachées. (Source de toutes les images de cet article : Elettronica In — https://futuranet.it)

données complexes. Ils sont composés d'unités de calcul appelées *neurones artificiels* ou *nœuds*. Ces neurones sont reliés entre eux par des connexions artificielles appelées *poids*. Chaque connexion est associée à une valeur numérique, qui représente l'importance de la connexion au modèle. Les réseaux neuronaux sont organisés en couches, avec une ou plusieurs couches qui sont cachées entre la couche d'entrée et la couche de sortie. La première couche, appelée *couche d'entrée*, reçoit les données d'entrée. Les couches intermédiaires, appelées *couches cachées*, traitent les informations par l'intermédiaire de leurs neurones. Enfin, la dernière couche, appelée *couche de sortie*, produit les résultats souhaités, comme le montre le schéma suivant à la **figure 1**.

Les réseaux neuronaux peuvent être utilisés dans un large éventail d'applications, telles que la traduction automatique, le traitement du langage naturel, le diagnostic médical, et dans ce qui noue intéresse ici, la reconnaissance d'images. Dans ce cas précis, au cours du processus d'apprentissage, des filtres sont appliqués à l'image à différentes résolutions, et la sortie de chaque image traitée est utilisée comme entrée pour la couche suivante. Les filtres commencent par des caractéristiques de base, comme la luminosité ou les bords, et deviennent de plus en plus complexes au fur et à mesure de leur évolution, en incluant des caractéristiques qui définissent l'objet de manière unique.

Les réseaux neuronaux convolutifs (CNN) sont un type de réseau neuronal spécifique, conçu pour le traitement efficace de données structurées, comme des images ou des vidéos. Ce qui distingue les CNN des réseaux neuronaux traditionnels, c'est l'utilisation d'une opération appelée *convolution*. Pendant la convolution, une petite fenêtre appelée *filtre* ou *noyau* glisse au-dessus de l'image d'entrée et une série d'opérations mathématiques est appliquée à chaque partie de l'image. Ce processus permet au CNN d'extraire automatiquement et efficacement les caractéristiques pertinentes des images, comme les bords, les textures ou les motifs. Ces réseaux se composent également d'une couche d'entrée, d'une couche de sortie et de nombreuses couches intermédiaires appelées couches *cachées*. Ils contiennent trois principaux types de couches, à savoir :

- La couche de convolution
- La couche de *Pooling*
- La couche complètement connectée (FC).

La couche convolutive est la première couche d'un réseau neuronal convolutif. Elle est chargée d'extraire les caractéristiques principales de l'image d'entrée. Les couches convolutives peuvent être suivies d'autres couches convolutives ou de couches de pooling. Les couches convolutives suivantes continuent de traiter les caractéristiques extraites des couches précédentes, ce qui permet au réseau d'apprendre des fonctionnalités de plus en plus complexes et abstraites.

La couche complètement connectée, également appelée couche de sortie, est la dernière couche d'un CNN. À ce niveau, les caractéristiques extraites sont utilisées pour effectuer des prédictions ou des classifications. Ce niveau est chargé de fournir la sortie finale du réseau neuronal à convolution.

À chaque niveau, la complexité du réseau neuronal convolutif augmente au fur et à mesure que le réseau apprend des caractéristiques de plus en plus sophistiquées et abstraites de l'image d'entrée. De plus, à mesure que l'on progresse dans les niveaux de convolution, la partie de l'image identifiée et analysée en détail par le réseau neuronal augmente, comme illustré à la **figure 2**. Contrairement à un réseau neuronal traditionnel, un CNN a des poids et des biais partagés qui sont les mêmes pour tous les neurones cachés dans une couche donnée. Après l'apprentissage des caractéristiques dans plusieurs couches, l'architecture d'un CNN passe à la classification.

L'avant-dernière couche est une couche entièrement connectée qui génère un vecteur de taille *K* (où *K* est le nombre de classes

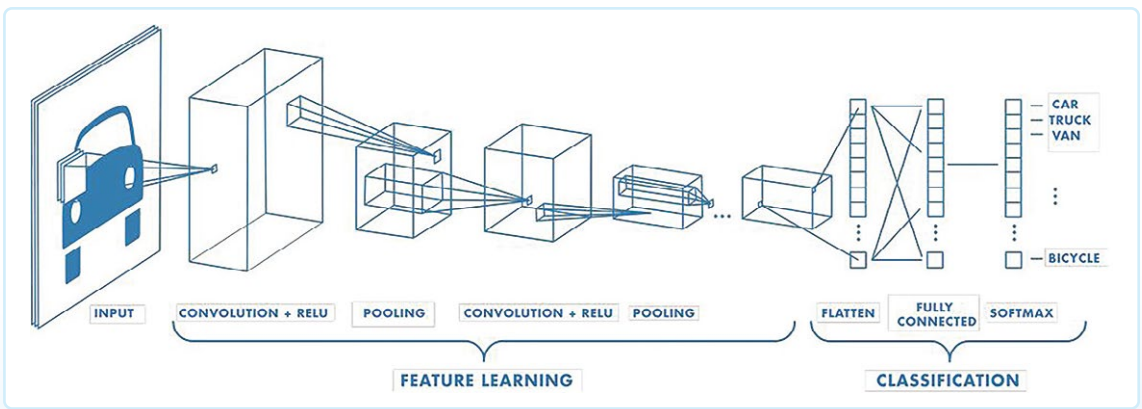

Figure 2. À chaque couche, la complexité du CNN augmente.

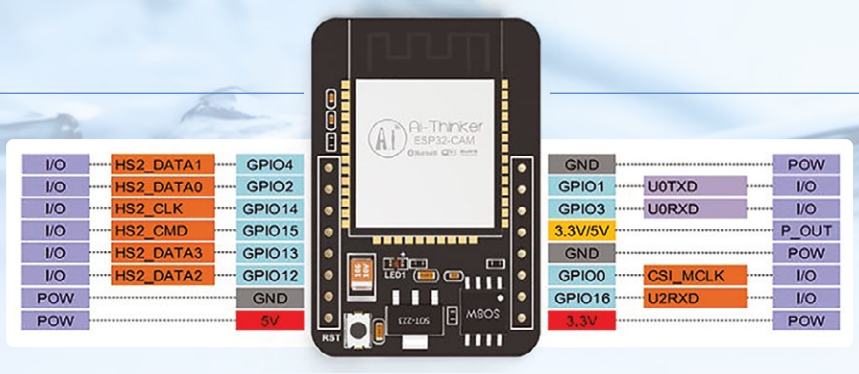

Figure 3. Brochage du module ESP32-CAM.

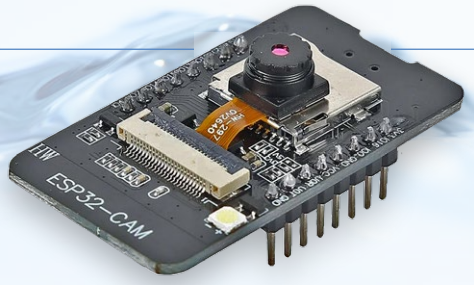

Figure 4. Le module ESP32-CAM est le cœur de ce projet.

prévisibles) et contient les probabilités pour chaque classe de toute image classée. La dernière couche de l'architecture CNN utilise une couche de classification pour fournir la sortie de la classification finale. En général, vous disposez de modèles pré-entraînés de ces réseaux qui pèsent évidemment plusieurs Mo ou Go. Dans notre cas, il s'agira de quelques Mo, mais ils seront sauvegardés sur une carte SD afin de ne pas surcharger la mémoire interne de la puce.

Il y a plusieurs fichiers présents dans le dossier config, plus précisément avec l'extension *tflite*, ce qui signifie Tensorflow Lite. TensorFlow, né de Google Brain en 2015, est devenu une bibliothèque de référence pour la création de modèles de Deep Learning. Initialement développée en interne par Google, elle a été mise en open source et a rapidement gagné en popularité au sein de la communauté de l'apprentissage automatique. Elle offre un large éventail de modèles et d'algorithmes d'apprentissage automatique et d'apprentissage profond, connus sous le nom de réseaux neuronaux, et les met à la disposition des développeurs avec une API intuitive.

TensorFlow exploite la puissance de Python ou de JavaScript pour fournir une interface de programmation facile à utiliser pour créer des applications. Parallèlement, l'exécution de ces applications se fait en C++, ce qui permet d'obtenir des performances de calcul élevées. Cela fait de TensorFlow un choix polyvalent pour les projets d'apprentissage automatique à grande échelle.

Les modèles entraînés avec TensorFlow peuvent également être mis en œuvre sur des appareils mobiles ou de périphérie, comme dans notre cas, ainsi que sur des systèmes d'exploitation iOS ou Android. L'écosystème TensorFlow propose des outils tels que TensorFlow Lite, qui optimise les modèles TensorFlow pour qu'ils fonctionnent efficacement sur ces appareils. Avec TensorFlow Lite, il est possible de faire un compromis entre la taille du modèle et sa précision. Un modèle plus petit peut occuper moins d'espace, par exemple 12 Mo au lieu de 25 Mo ou même plus de 100 Mo, mais il peut présenter une légère perte de précision. Cependant, cette perte de précision est souvent négligeable, compte tenu des avantages en termes de vitesse et d'efficacité énergétique qu'offre le modèle compressé. Dans notre cas ici, ce modèle entraîné avec TensorFlow sera utilisé pour distinguer et reconnaître les nombres dans le compteur et pour déterminer la direction des indicateurs. Avec l'aide de TensorFlow, nous pourrons optimiser la puissance de l'apprentissage profond pour l'analyse des données et l'IA, en améliorant l'efficacité et la précision de nos applications.

Matériel

Comme nous l'avons mentionné, le module choisi est l'ESP32-CAM d'AI-Thinker qui, grâce à sa taille compacte (40,5 × 27 × 4,5 mm), pourrait être la crème de la crème de tous les *makers*. Succédant au célèbre ESP8266, dont nous parlerons plus loin, il se compose d'un module ESP32 équipé d'un connecteur, dont le brochage est visible sur la **figure 3,** pour accueillir un module caméra séparé et un emplacement pour une carte SD d'une capacité maximale de 4 Go. Dans le détail, nous pouvons voir un microcontrôleur programmable avec Wifi et Bluetooth intégrés, et une mémoire RAM externe supplémentaire pouvant aller jusqu'à 4 Mo.

En plus de ça, le nouveau connecteur de caméra peut intégrer un module OV2640 ou OV7670, le premier est intégré au module lui-même et a une résolution de 2 mégapixels. Sa vitesse SPI est de 8 MHz et la taille de la mémoire tampon est de 384 Ko. Sa consommation normale est de 70 mA, tandis qu'en mode économie d'énergie, elle n'est que de 20 mA, ce qui la rend également intéressante pour sa fonction de faible consommation d'énergie.

Il y a également une LED à haute luminosité sur la carte (**figure 4**), qui peut servir de flash ou d'éclairage de la scène à filmer. Cette caractéristique est essentielle pour notre projet, puisqu'il sera placé dans des environnements qui sont normalement très sombres. Nous verrons qu'il sera possible de faire varier cette lumière pour obtenir le résultat souhaité. Bien entendu, si cela ne suffit pas, les nombreux GPIO montés sur la carte nous permettent éventuellement de monter une source lumineuse externe en cas de faible luminosité.

Pour les petites applications, cette puce se révèle très robuste et dotée d'une grande puissance de calcul, puisqu'elle s'appuie sur 2 cœurs 32 bits à 120 MHz. Cette dernière caractéristique lui permet d'avoir un taux de rafraîchissement assez élevé, bien sûr en fonction du format et de la taille. À titre indicatif, nous pouvons atteindre un débit de 8 JPEG en SVGA (800x600) par seconde.

La programmation de notre module ESP32-CAM peut se faire de plusieurs manières, la plus classique étant l'utilisation d'un adaptateur de FTDI **(figure 5**), qui grâce à l'interface FT232RL, simule les ports RS-232 et COM et permet ainsi un *plug and play* rapide du module.

Note de l'éditeur : Le module suggéré dans l'encadré « Produits associés » à la fin de cet article, entièrement compatible avec les broches et disponible dans la boutique Elektor, intègre un connecteur micro-USB avec l'interface série correspondante au chipset, ce qui rend inutile tout adaptateur série externe.

Si vous ne disposez pas de l'objet décrit ci-dessus, ne désespérez pas. En fait, il est possible de le faire à partir de n'importe quel module dans lequel les broches TX et RX sont présentes, donc à partir de différents modèles d'Arduino ou d'ESP.

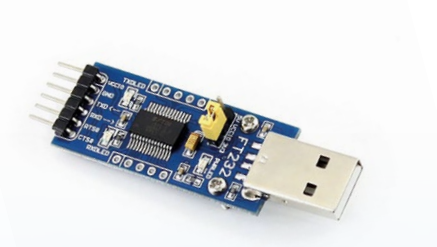

Figure 5. Le convertisseur USB/TTL avec FT232RL.

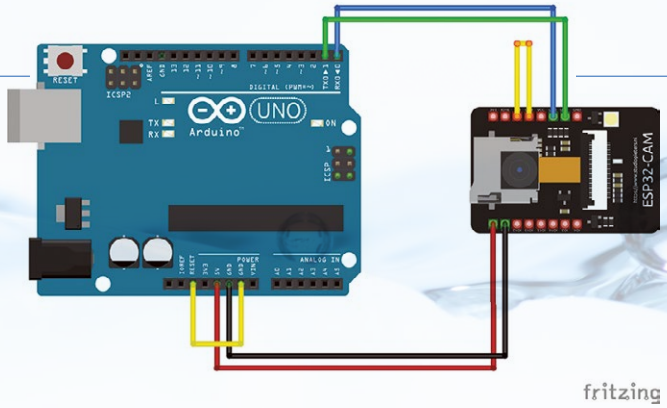

Figure 6. Le module ESP32-CAM peut être programmé avec un Arduino UNO.

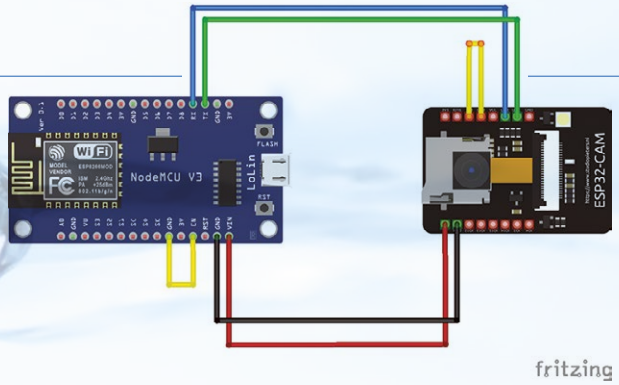

Figure 7. Câblage pour programmer le module ESP32-CAM avec un ESP8266.

Tableau 1. câblage de l'Arduino à l'ESP32-CAM.

Arduino ⇒ ESP32-CAM	Arduino	ESP32-CAM
TX ⇒ U0TXD	Reset ⇒ GND	GPIO0 ⇒ GND
RX ⇒ U0RXD		
5V ⇒ 5V		
GND ⇒ GND		

Tableau 2. Câblage de l'ESP8266 à l'ESP32-CAM.

ESP8266 ⇒ ESP32-CAM	ESP8266	ESP32-CAM
TX ⇒ U0TXD	EN ⇒ GND	RX ⇒ U0RXD
5V ⇒ 5V		
GND ⇒ GND		

Câblage pour programmer le module ESP32-CAM

Pour la raison mentionnée ci-dessus, il peut y avoir différents types de câblage, similaires mais différents à certains égards. En général, le module ESP32-CAM n'a pas de port USB pour pouvoir se connecter directement au PC, et c'est précisément pour cette raison que nous devrons utiliser un module externe.

Commençons par le plus classique des modules, un Arduino UNO. Dans un premier temps, nous identifions les broches d'émission et de réception de notre ESP32-CAM, appelés respectivement U0TXD et U0RXD. Ces broches devront être connectées directement aux broches TX et RX de notre Arduino. Dans le module de la caméra, ces broches correspondent également aux broches GPIO1 et GPIO3. Ensuite, nous court-circuitons les contacts GND et IO0 de l'ESP32-CAM pour assurer une communication correcte. Pour l'alimentation, nous connectons en parallèle les broches 5V et GND des deux cartes

Enfin, pour réinitialiser l'Arduino si nécessaire, nous court-circuitons la broche RESET et la broche GND. En résumé, nous obtenons le diagramme de la **figure 6**, également illustré dans le document **tableau 1**.

Si nous n'avons pas un Arduino UNO mais plutôt un ESP8266, le câblage change mais très peu. En fait, le seul changement qui mérite d'être mentionné concerne les contacts à court-circuiter sur notre module programmateur. Il ne s'agit plus de RESET et GND mais de EN et GND. Le branchement est illustré dans la **figure 7** et indiqué dans le **tableau 2** également.

Et pour finir, le câblage le plus simple de tous. En fait, si nous possédons un module convertisseur USB/TTL basé sur la puce FTDI, tout est déjà réglé avec une connexion de moins. En effet, du côté du programmateur, nous n'aurons plus à court-circuiter quoi que ce soit, mais nous devrons échanger les contacts TX et RX l'un avec l'autre, comme le montrent la **figure 8** et le **tableau 3**.

Les connexions évoquées ne sont nécessaires que pendant la phase de programmation de l'ESP32-CAM. Une fois la programmation terminée, il suffit d'alimenter le module ESP32-CAM en utilisant uniquement les fils d'alimentation 5V et GND. Il n'est plus nécessaire de le connecter directement à l'Arduino ou d'effectuer d'autres câblages.

Dans la deuxième et dernière partie de cet article, nous verrons la procédure d'installation du micrologiciel, la configuration correcte de l'objectif de la caméra pour une mise au point optimale, le positionnement du lecteur sur le compteur d'eau, et bien évidemment, l'ensemble du processus basé sur l'IA pour la reconnaissance et la lecture correctes des éléments du compteur. Restez à l'écoute !

VF : Laurent Rauber — 240213-04

Tableau 3. Câblage du convertisseur FTDI vers l'ESP32-CAM.

FTDI	ESP32-CAM
RX	U0TXD
TX	U0RXD
5V	5V
GND	GND

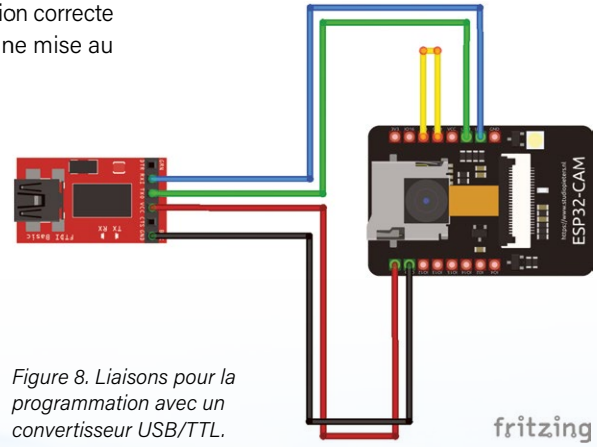

Figure 8. Liaisons pour la programmation avec un convertisseur USB/TTL.

Questions ou commentaires ?

Contactez Elektor (redaction@elektor.fr).

Produits

- **ESP32-Cam-CH340 Carte de développement**
  www.elektor.fr/19333
- **FTDI Serial TTL RS232 USB Cable**
  www.elektor.fr/20173