intelligence artificielle pour débutants (2)

Réseaux de neurones avec Linux et Python

Walter Trojan (Allemagne)

Le premier article de cette petite série [1] présentait le matériel Maixduino et montrait comment le programmer en C++ avec l'IDE Arduino. La puissance du processeur a été démontrée par un modèle d'IA pour la reconnaissance d'objets. Cet article portera sur le fonctionnement de l'intelligence artificielle dans l'apprentissage (ap)*profond*(i). Si vous voulez vous frotter à l'IA, Linux et Python sont essentiels. Pour que ça reste un parcours de santé, il suffit de choisir les outils appropriés.

N'escomptez pas atteindre la maîtrise de l'apprentissage approfondi à la lecture de cet article. J'ai moi-même étudié quelques livres et suivi nombre de tutoriels puis fait de nombreuses tentatives de programmation et il me reste beaucoup à apprendre. Je voudrais au moins vous donner un aperçu des structures et des méthodes utilisées pour équiper d'une intelligence spécialisée un programme universellement applicable sous la forme d'un réseau neuronal (NN).

Constitution d'un réseau neuronal

Un réseau neuronal (que nous désignerons NN dans la suite de cet article) est constitué de plusieurs couches qui ont chacune plusieurs nœuds. On se représente les nœuds d'une couche (**fig. 1**) disposés verticalement. Théoriquement, le nombre de nœuds par couche et le nombre de couches dans le réseau peuvent être quelconques. L'architecture est déterminée par la tâche concernée et la taille dépend bien sûr aussi des ressources de la plateforme informatique utilisée. Un exemple : si le NN doit classer des objets qui lui sont transmis sous forme d'images, la première couche prend en charge les données pour résoudre la tâche et fournit un nombre correspondant de nœuds d'entrée pour l'acquisition. Si une image à faible résolution ne comporte que 28 * 28 pixels, il faudra donc 784 nœuds pour un affichage en valeurs de gris. Pour l'acquisition d'images en couleur, ce serait le triple. Lors de l'acquisition d'une image, chaque nœud d'entrée reçoit la valeur de gris de « son » pixel.

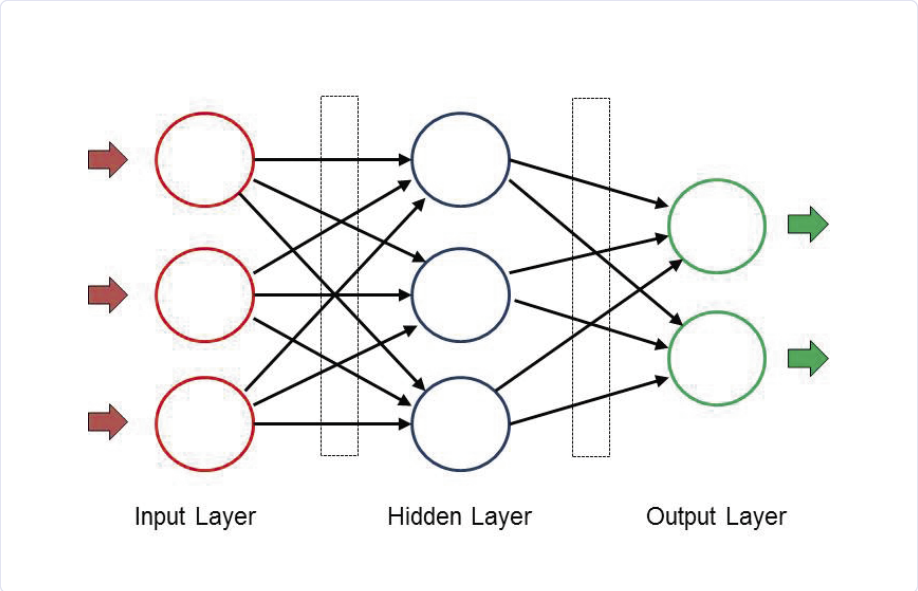

Figure 1. Structure d'un réseau neuronal.

Le résultat de l'analyse des images est présenté dans la couche de sortie, qui comporte un nœud distinct pour chaque résultat. Si le NN reconnaît 1000 objets, cette couche aura le même nombre de nœuds de sortie. Chaque nœud présente une probabilité comprise entre 0 et 1,0 et indique ainsi le degré de fiabilité du résultat trouvé par le NN. Exemple : lorsque le résultat est ***chat domestique***, le nœud associé pourrait avoir une valeur de 0,85 et le nœud ***tigre*** une valeur de 0,1. Le degré de fiabilité des autres nœuds serait encore plus explicite et le résultat serait clair.

Au cours de l'apprentissage automatique, l'analyse proprement dite est assurée par des couches cachées. Selon la tâche, un nombre quelconque de couches cachées peut être utilisé ; en pratique, il y a facile-

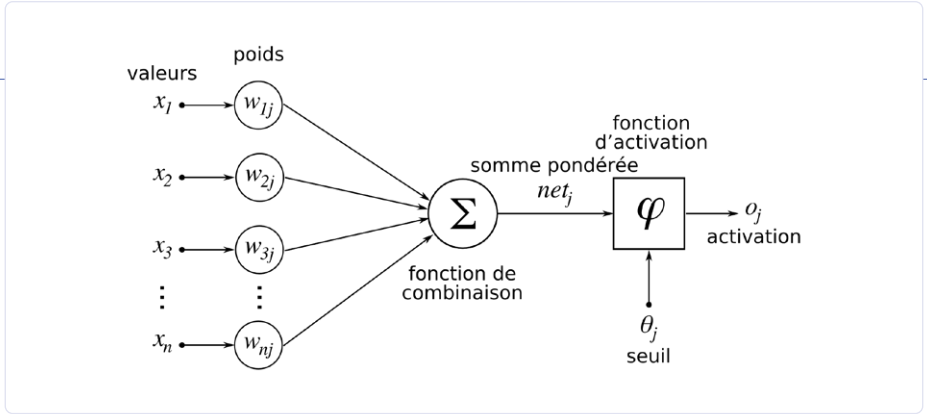

Figure 2. Calculs à l'intérieur d'un nœud (source : https://commons.wikimedia.org/wiki/Artificial_neural_network).

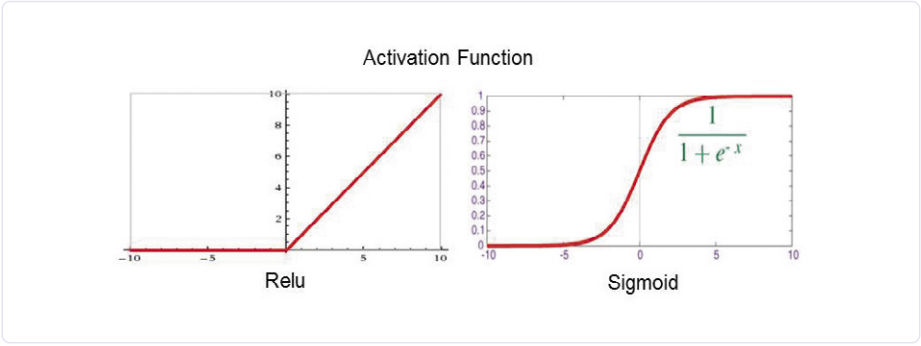

Figure 3 Fonctions d'activation fréquemment utilisées.

ment 100 à 200 de ces couches en fonction. L'architecture présentée ci-dessus n'est pas la seule, on trouve aussi des structures plus complexes avec rétroaction entre couches ou des filtres intermédiaires pour affiner la précision. Le nombre de nœuds dans chaque couche cachée est également arbitraire. Dans l'exemple mentionné pour la détection d'objets, trois couches cachées de 200 nœuds chacune pourraient déjà fournir des résultats potables.

Dans le cerveau humain, l'intelligence naît du couplage de neurones par l'intermédiaire des synapses ; les nœuds du NN sont interconnectés eux aussi. Chaque nœud d'une couche est connecté à tous les nœuds des couches suivantes. Chaque connexion (représentée par une flèche dans l'image) contient une valeur appelée *poids*. Ces poids entre couches sont stockés dans des matrices. Ce qui explique la prédilection de l'apprentissage automatique pour les langages de programmation qui facilitent les opérations matricielles rapides.

Comment un NN obtient-il les résultats souhaités ? Chaque nœud reçoit ses signaux d'entrée de tous les nœuds de la couche située devant lui, marquée x sur la **figure 2.** Ces valeurs sont multipliées par les valeurs de poids **w** et additionnées, de sorte que l'entrée du réseau (*net* en anglais)

net = x1*w1 + x2*w2 + X3*w3 ...

La valeur calculée de *net* est maintenant multipliée par une fonction d'activation et envoyée à la sortie (et donc à tous les nœuds de la couche suivante). Une valeur de seuil, spécifique au nœud, détermine également son **activation**.

Les fonctions d'activation illustrées par la **figure 3** sont attribuées aux couches et garantissent que les valeurs de sortie restent dans la plage souhaitée. Par exemple, la fonction Relu supprime toutes les valeurs négatives tandis que Sigmoid les limite entre 0 et 1. Grâce à ces précautions, les NN fonctionnent dans une plage numérique définie et ne peuvent pas être dominés par des «valeurs aberrantes».

Formation

Ainsi, après l'enregistrement des données d'entrée, NN effectue de nombreux calculs dans chaque couche et présente les résultats à la sortie. Ce processus est appelé inférence. Vous vous demandez comment il peut être question d'intelligence dans tout cela... Comme chez un enfant, par l'instruction et la formation. Pour un NN sans instruction, les *poids* sont généralement des nombres aléatoires dans la plage de –1 à +1, le réseau est ignorant et ne fournit donc que des résultats aléatoires. Pour le former, les données et les valeurs cibles de sortie connues sont transmises au réseau. Après calcul avec ces valeurs, le résultat est comparé au point de consigne et la différence documentée par une fonction de perte. Suit maintenant le processus d'apprentissage, également appelé *rétropropagation*. À partir de la dernière couche, les poids sont ajustés par petits pas afin de réduire la perte au minimum. Après de nombreux cycles d'entraînement, souvent des millions, avec différentes données d'entrée, tous les poids finissent par avoir les valeurs appropriées à la tâche ; notre NN studieux peut maintenant analyser de nouvelles données qui lui sont inconnues.

Une architecture bien adaptée au traitement de l'image et du son est le *réseau neuronal convolutif*, reconnu par Maixduino. Dans un tel CNN (*Convolutional Neural Network*), les neurones sont (au moins dans certaines couches) disposés en deux dimensions, ce qui correspond bien aux données d'entrée bidimensionnelles comme celles d'une image. Par rapport au réseau de la figure 1, où l'activité d'un neurone dépend de tous les neurones de la couche précédente (via différents facteurs de pondération), la dépendance dans le cas du CNN est simplifiée et localement limitée. Ici, l'activité d'un neurone ne dépend que des valeurs de (par exemple) 3 x 3 neurones situés dans la couche située devant lui – les facteurs de pondération sont les mêmes dans ce cas. Un tel réseau reconnaît particulièrement bien les petites structures telles que les lignes, les courbes, les points et motifs similaires. C'est aux couches suivantes qu'il appartient de reconnaître des détails plus complexes et finalement des visages entiers.

En termes de programmation, un NN est comparable à un tableur : un arrangement de cellules avec des instructions de calcul prédéfinies, qui pendant l'exécution accède avec les poids à des matrices multidimensionnelles. Lors de la classification de nouvelles données d'entrée, toutes les couches de l'entrée à la sortie sont parcourues et les sorties reflètent le degré de probabilité du résultat attribué au nœud. Les NN bien entraînés peuvent atteindre des valeurs de l'ordre de 0,9. Pendant l'entraînement, après l'interférence, la rétropropagation a lieu, en sens inverse, de la sortie vers l'entrée pour ajuster les poids.

Tout ceci reste un peu compliqué au début, j'en conviens, mais on verra que pour la mise en œuvre il existe de nombreux outils et bibliothèques que je présenterai dans le prochain article.

Linux et Python

Dès le premier article de cette petite série, j'ai laissé entendre que pour aborder l'IA, il faudrait fournir un effort. Le meilleur endroit du monde pour s'y mettre est la plateforme Linux, où la plupart des outils sont gratuits et de bonne qualité. Linux offre le même confort que Windows, mais il est conditionné différemment, généralement mieux. Ma préférence va à Ubuntu, également disponible en version LTS (*Long Term Support*) supportée au moins 4 ans. Les autres dérivés de Linux comme Debian, Mint etc. sont tout aussi utilisables, le choix est une question de goût. Linux peut être installé sur une machine virtuelle en plus de Windows, vous n'avez donc pas besoin d'un ordinateur supplémentaire.

Et pourquoi Python ? Surtout si c'est un langage de programmation interprété, donc potentiellement lent diront certains. Je trouve cet inconvénient largement compensé par de nombreux avantages : tout d'abord, Python se passe d'accolades, points-virgules et autres fioritures. C'est l'indentation des lignes qui structure les blocs. Il offre de puissantes structures de données telles que listes, tuples, ensembles et dictionnaires ; de plus, Python dispose du calcul matriciel intégré. Les autres avantages majeurs sont les infrastructures et bibliothèques d'intelligence artificielle disponibles, qui, grâce à leur grande modularité, peuvent être intégrées totalement ou partiellement. Écrites en C++, elles offrent donc les performances requises. Quelques instructions suffironnt pour installer tout cela sans difficultés.

Pour vous lancer, installez le Linux de votre choix, ainsi que *pip3* et Python 3. Vous trouverez sur la toile les instructions idoines.

Maixduino parle MicroPython

Pour permettre à Python de fonctionner sur des systèmes moins riches en mémoire, il en existe la version *MicroPython* allégée, qui peut être installée sur des plateformes comme Maixduino, ESP32 et d'autres. Outre son solide jeu de commandes, MycroPython offre 55 modules supplémentaires pour de nombreuses fonctions pour les mathématiques et pour le système. Pour y ajouter de nouvelles versions ou des modèles d'IA, il vous faudra l'outil `Kflash`.

Installation de Kflash sous Linux :

- Téléchargez la version 1.5.3 ou supérieure [2] compressée *kflash_gui_v1.5.3_linux.tar.xz*
- Transférez-la vers un dossier de votre choix
- Décompressez avec la commande `tar xvf kflash_gui_v1.5.3_linux.tar.xz`
- Passez au dossier nouvellement créé */ kflash_gui_v1.5.2_linux/kflash_gui*
- Commencez par *./kflash_gui* (en cas de problèmes de démarrage, cochez dans les propriétés de ce fichier la case *Exécuter comme programme*).

L'interface graphique de Kflash (**Fig. 4**) est ainsi lancée et le micrologiciel ou les modèles d'IA peuvent maintenant être chargés dans Maixduino.

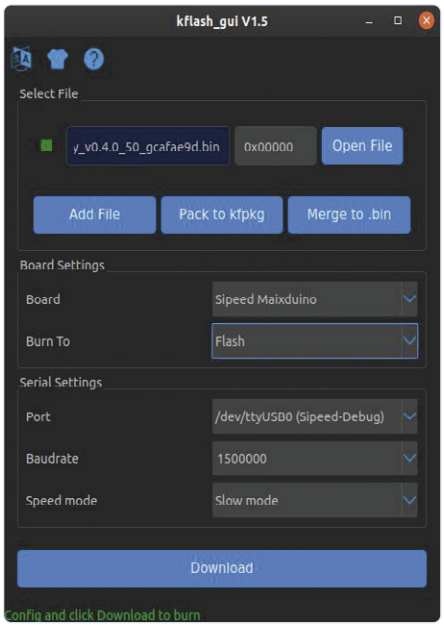

Figure 4. Interface graphique de Kflash.

Installation d'un micrologiciel sur Maixduino

Même si Maixduino est livré équipé de MicroPython, il est recommandé d'en télécharger la dernière version. Au moment de rédiger cet article (mai 2020), le micrologiciel est v0.5.0 [3]. Choisissez *maixpy_v0.5.0_8_g9c3b97f* ou supérieur et dans l'image suivante choisissez la variante *maixpy_v0.5.0_8_g9c3b97f_minimum_with_ide_support.bin* ou supérieur, un fichier d'environ 700 Ko qui contient également le support pour l'IDE MaixPy.
Installez rapidement le nouveau micrologiciel avec Kflash. Après avoir réglé la *carte*, le *port*, le *débit en bauds* et le *mode de vitesse* (**fig. 5**), cliquez sur *Télécharger* pour lancer le téléchargement. Après cela, vous pouvez déjà exécuter sur le Maixduino via le port */dev/ttyUSB0* les premières commandes Python avec un émulateur de terminal (p. ex. Putty). Voici un petit exemple avec des commandes de tableau (*array*) :

```
>>>                                  # invite Python
>>> import array as arr              # importer le module Array
>>> a = arr.array('i',[1,2,3])       # créer un tableau a d'entiers
>>> b = arr.array('i',[1,1,1])       # créer un tableau b d'entiers
>>> c = sum(a + b)                   # additionner les valeurs du tableau
>>> print(a,b,c)                     # et les afficher
array('i', [1, 2, 3]) array('i', [1, 1, 1]) 9     # affichage
>>>
```

Grâce aux bibliothèques comme *numpy* (sous Linux) ou *umatlib,* d'autres fonctions intéressantes sont disponibles.

Installation de l'IDE MaixPy

La programmation est confortable grâce à l'environnement de développement MaixPy. Les programmes Python peuvent être écrits et testés puis chargés et exécutés sur Maixduino. En outre, il existe des outils d'analyse d'images (**fig. 5**). L'installation se déroule comme suit :

- Téléchargez la version *maixpy-ide-linux-x86_64-0.2.4-installer-archive.7z* ou supérieure [4]
- Transférez-la vers un dossier de votre choix
- Décompressez-la avec la commande *tar maixpy-ide-linux-x86_64-0.2.4-installer-archive.7z*
- Passez au nouveau dossier *maixpy-ide-linux-x86_64-0.2.4-installer-archive* et

saisissez les commandes suivantes :

```
./setup.sh
./bin/maixpyide.sh
```

L'IDE démarre. Pour les démarrages ultérieurs, seule la dernière commande sera nécessaire. À présent, tous les outils de mise en œuvre des modèles d'IA sont en place. Voyons si nous parvenons à reconnaître des visages.

Qu'est-ce qu'elle a…

On utilise un modèle d'IA formé pour la reconnaissance des visages, qui a déjà analysé les caractéristiques de plusieurs milliers de visages, et dont les poids des NN utilisés sont ajustés en conséquence. Le modèle est disponible [5] sous le nom *face_model_at_0x300000.kfpkg. La* base du développement est l'infrastructure d'intelligence artificielle Yolo2 (*You Only Look Once*), qui divise les éléments de l'image en plusieurs zones, les analyse séparément et atteint ainsi des taux de reconnaissance élevés (je reparlerai des infrastructures d'IA plus tard dans cette série). Le modèle d'IA pour le processeur est emballé au *format kfpkg* et doit être flashé à l'adresse 0x300000 dans le Maixduino. Cette opération est possible aussi avec Kflash ; il suffit de trouver le fichier en utilisant *Open File* et de le charger sur la carte avec les paramètres adéquats (**fig. 4**).
Avec l'IDE MaixPy, la manipulation du script Python est confortable. Il facilite l'écriture et la mise au point des programmes et leur transfert vers Maixduino. L'interface comporte trois fenêtres (**fig. 5**) :

- **Éditeur**, en haut à gauche, pour la saisie de programmes avec affichage de la syntaxe.
- **Console**, en bas, pour l'affichage de la sortie du programme.
- **Analyse**, à droite, avec la division spectrale des images en rouge, vert et bleu.

Parmi les boutons et options de menu disponibles, deux sont importants. Le trombone permet de se connecter (vert) ou de se déconnecter (rouge) au port `ttyUSB0` de Maixduino. Le triangle vert en dessous lance le script ; ce bouton se transforme en un point rouge avec un «x» et sert à arrêter le programme.
Pour essayer, j'ai imprimé deux visages connus (Albert Einstein et Rudi Völler) choisis au hasard et les ai épinglés au mur. On m'a dit que ces deux visages présentaient une certaine ressemblance, mais je ne la vois pas. Ces visages ont immédiatement été reconnus comme tels et marqués par un cadre. Veillez à respecter le format paysage, sinon le taux de reconnaissance diminue sensiblement.

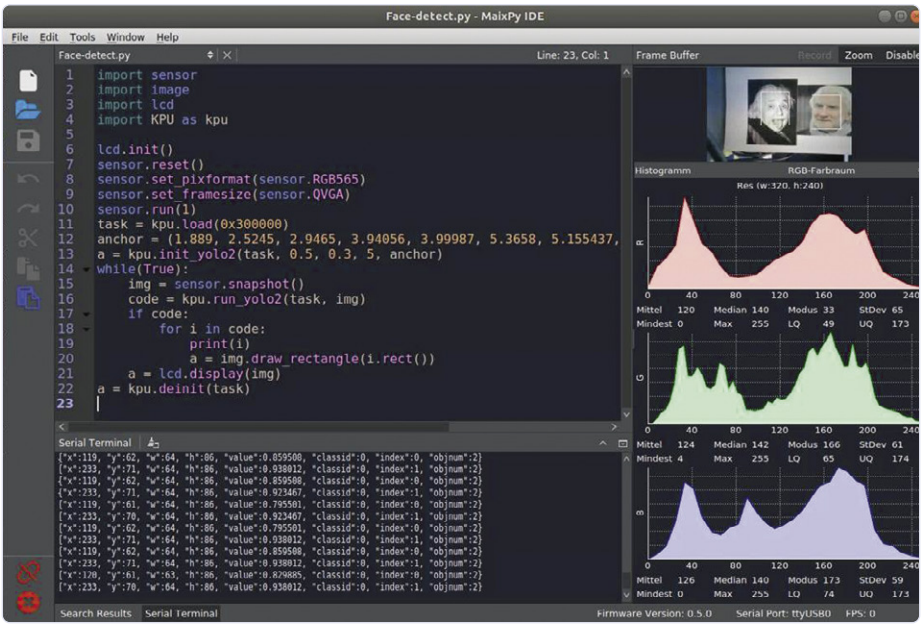

Figure 5. Interface de l'IDE MaixPy.

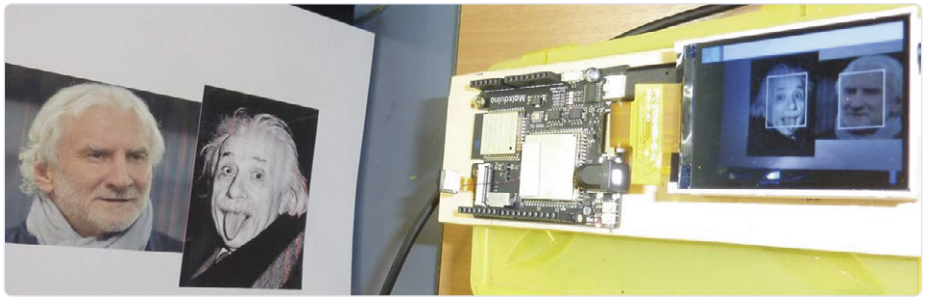

Figure 6. Mon dispositif de test pour la reconnaissance des visages.

Le programme *Face-detect.py se* trouve dans le dossier de téléchargement sur le site d'Elektor [6]. Sa brièveté témoigne de l'excellence des bibliothèques utilisées. Pour commencer, les bibliothèques nécessaires pour l'appareil photo, l'écran LCD et la KPU sont intégrées et initialisées. Ensuite, le NN est chargé dans la KPU à partir de l'adresse 0x300000. Lorsque le modèle d'IA est initialisé par la commande `kpu.init_yolo2`, des constantes supplémentaires sont transférées pour le réglage de la précision et de l'optimisation. Puis la classification des images a lieu dans une `boucle` sans fin, une image est prise et envoyée au NN. Si des visages ont été détectés, la variable `i` reçoit pour chaque visage les coordonnées et la taille d'un cadre marqueur, superposé ensuite à l'image. Puis apparaissent l'image (sur l'écran LCD) et les données de marquage (sur la console). Pour plus de détails sur les commandes de la KPU, voir le lien [7].
Dans l'IDE MaixPy, l'image est également affichée dans le coin supérieur droit et son histogramme des couleurs en dessous. Si vous ne voulez pas ces informations, désactivez la fenêtre de droite.

Pour faciliter les manipulations, j'ai monté le tout sur une plaque (**fig. 6**) et orienté la caméra pour faciliter la capture de visages réels, d'images imprimées ou de contenus d'écran pour les analyser.
La **fig. 7** montre ce que ça donne sur l'écran LCD. Aucune similitude entre ces deux visages n'a été signalée.
Voyons ce que cache ce modèle Yolo2 ? Le réseau neuronal comporte 24 couches de

Figure 7. Voici ce que ça donne sur l'écran LCD de Maixduino.

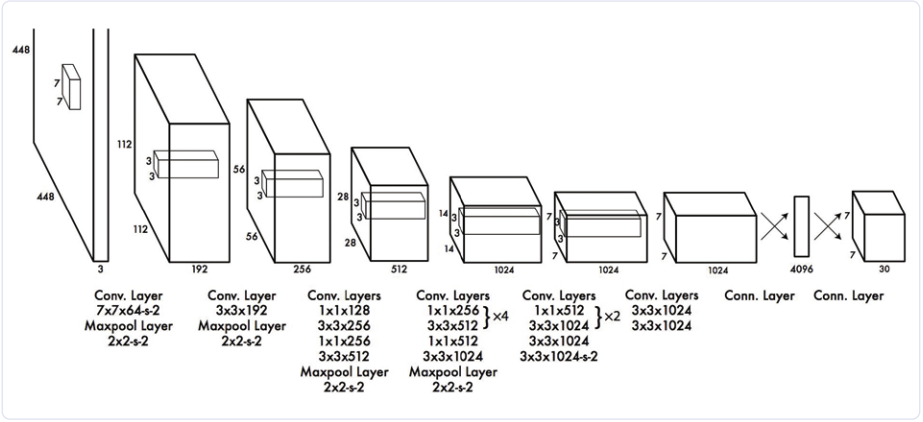

Figure 8. Architecture de réseau pour la reconnaissance des visages (source : https://bit.ly/3cK3DUR).

convolution et deux couches de sortie entièrement connectées (**fig. 8**). Entre les deux, certaines couches de *maxpools* (les *pools* [8] sont des neurones de mise en commun) sont interposées comme filtres pour éliminer de la complexité et pour atténuer la tendance à la « mémorisation ». On constate que la fréquente utilisation de la taille de fenêtre de 3x3 pour la reconnaissance des détails. C'est exactement ce que permet le matériel de la KPU, qui garantit que le Maixduino est très efficace pour de telles tâches.
D'autres structures NN connues comptent jusqu'à plusieurs centaines de couches, sont dotées de régressions ou d'autres fonctions complémentaires. Les limites de la créativité dans ce domaine ne sont qu'une question de puissance de calcul, donc de budget.

… ma gueule ?

Le matériel puissant et l'environnement logiciel disponible avec Maixduino montre à quel point c'est l'outil idéal pour faire ses premiers pas dans l'Intelligence Artificielle. Sa faible consommation permet l'utilisation sur des appareils mobiles de réseaux neuronaux déjà formés. Je montrerai dans le prochain épisode comment développer, former et exécuter votre propre réseau de neurones. L'interface avec le développeur sera l'infrastructure d'intelligence artificielle Keras, appelé à raison le Lego® de l'IA. Nous verrons aussi comment programmer l'ESP32, p. ex. pour l'acquisition de valeurs analogiques.

Votre curiosité sera le moteur de vos progrès pour lesquels vous seront bénéfiques certaines lectures complémentaires. [8][9]

200023-B-02

LIENS

[1] **intelligence artificielle pour débutants (1), Elektor mai/juin 2020, p.12** : www.elektormagazine.fr/magazine/elektor-148/58637
[2] **Kflash** : https://github.com/sipeed/kflash_gui/releases
[3] **micrologiciel Maixduino** : http://dl.sipeed.com/MAIX/MaixPy/release/master/
[4] **IDE MaixPy** : http://dl.sipeed.com/MAIX/MaixPy/ide/_/v0.2.4/maixpy-ide-linux-x86_64-0.2.4-installer-archive.7z
[5] **modèles d'IA** : http://dl.sipeed.com/MAIX/MaixPy/model
[6] **logiciel** : www.elektormagazine.fr/200023-B-02
[7] **commandes KPU** : https://maixpy.sipeed.com/en/libs/Maix/kpu.html
[8] **réseau neuronal convolutif** : https://fr.wikipedia.org/wiki/R%C3%A9seau_neuronal_convolutif#Couche_de_pooling_(POOL)
[9] **neurone formel** : https://fr.wikipedia.org/wiki/Neurone_formel