carte spéciale IA

apprentissage automatique avec le Jetson Nano

Tam Hanna (Hongrie)

L'intelligence artificielle sur les GPU n'est pas nécessairement synonyme de cartes graphiques coûteuses et de consommation d'énergie énorme. NVIDIA s'oriente depuis un certain temps vers des systèmes plus petits, notamment en raison de la concurrence avec les fabricants de microcontrôleurs. Dans cet article, nous examinons le Jetson Nano et une petite application de démonstration.

L'accélération de l'intelligence artificielle (IA) dans les systèmes embarqués est un marché en pleine évolution. Des entreprises telles que Canaan et Maxim Integrated se livrent une lutte acharnée pour la suprématie. ARM s'aventure également sur ce terrain avec l'annonce du Cortex M52. L'objectif est de proposer de petits systèmes IA qui exécutent des tâches d'IA de base en périphérie, sans connexion à Internet. L'architecture système est ainsi beaucoup plus stable car les tâches de ML ou d'IA peuvent être poursuivies même si la connexion au serveur principal est perdue. NVIDIA essaie de se positionner dans ce domaine depuis un certain temps avec la série Jetson.

Par souci de transparence, il convient de noter que les cartes personnalisées de ces systèmes ne sont pas vraiment réalisables pour les petites entreprises, ne serait-ce qu'en raison de l'extrême largeur de bande nécessaire pour accéder à la mémoire vive DDR. La société NVIDIA est consciente de ce problème. Son aperçu de portefeuille (disponible sur [1]), qui donne aux électroniciens une vue d'ensemble de tout l'écosystème Jetson, comprend donc plusieurs « cartes de calcul », telles que le Jetson TX2 présenté dans **la figure 1**.
Il convient également de noter qu'avec la prédominance de NVIDIA dans le secteur des GPU, l'écosystème tiers bien développé a également découvert la gamme de produits Jetson. Une vue d'ensemble de divers produits tiers est disponible à l'adresse [2], dont l'intégration dans une solution interne peut faire gagner beaucoup de temps de développement (pensez, par exemple, à la conception du boîtier et à la sélection des caméras, etc.).

Démarrage du système et fonctionnement

Pour les amateurs, NVIDIA propose le kit de développement NVIDIA Jetson Nano présenté dans la **figure 2** (à côté d'un Raspberry Pi et d'un Orange Pi 5+). Au moment de la rédaction de cet article,

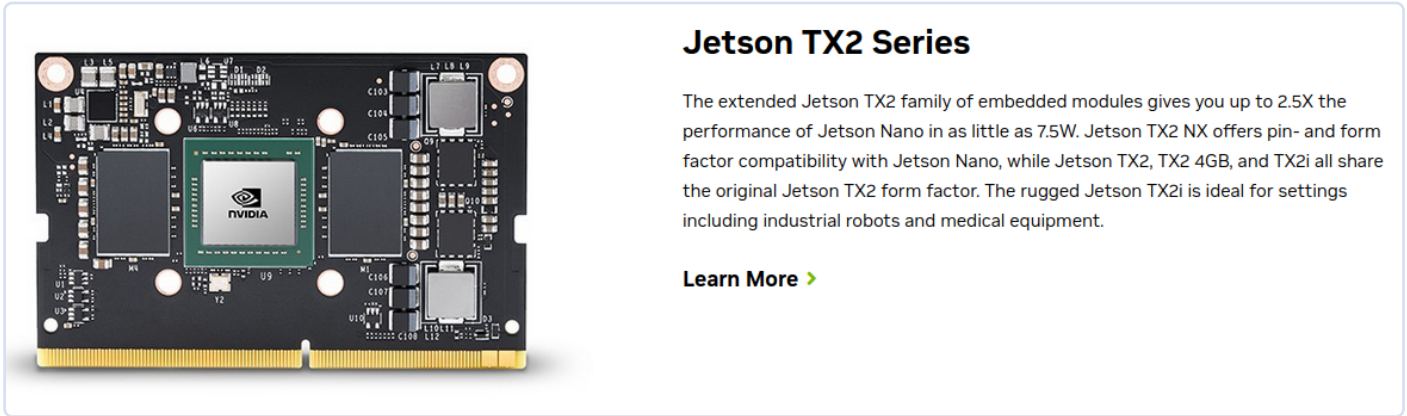


Figure 1. L'utilisation de ce module prêt à l'emploi facilite grandement l'intégration du Jetson dans des circuits propriétaires (source : NVIDIA).

le meilleur prix d'OEMSecrets (voir [3]) était de 155 €. Bien que légèrement plus cher qu'un Raspberry Pi, la technologie NVIDIA est mieux adaptée à l'écosystème général de l'IA. Lors de l'achat du NVIDIA Jetson Nano, il est recommandé de commander également un bloc d'alimentation avec un connecteur d'alimentation aux dimensions habituelles de 5,5/2,1 mm. Dans les étapes suivantes, l'auteur utilise un MeanWell GST25E05-P1J.
Un examen attentif du système (voir aussi la **figure 3**) montre qu'il s'agit d'un produit en deux parties. Outre la carte de support, qui contient les différentes interfaces, on trouve le module de calcul lui-même avec le dissipateur thermique. Il est nécessaire d'insérer une carte microSD contenant le système d'exploitation dans le module de calcul.

Micro-USB est également possible en théorie

Si vous disposez d'une alimentation Micro-USB très puissante et que vous n'utilisez pas le port Micro-USB pour connecter du matériel externe, vous pouvez également choisir cette méthode d'alimentation. Cependant, en raison de « conflits » avec le Raspberry Pi, l'auteur, comme un enfant brûlé, redoute le feu et se contente d'une alimentation CC classique.

Le SoC intégré est un système multicœur doté de quatre cœurs Arm Cortex A57, en plus de l'accélérateur d'IA lui-même. Il en résulte une configuration qui rappelle les ordinateurs de traitement classiques fonctionnant généralement sous Linux embarqué. NVIDIA recommande d'utiliser une carte MicroSD d'une capacité d'au moins 32 Go et d'une vitesse minimale de UHS1. Le fichier image est disponible à l'adresse [4], vous pouvez l'extraire sur votre carte mémoire avec un lecteur de carte comme à l'accoutumée.
Vous aurez également besoin d'une souris et d'un clavier USB pour le démarrage initial ; le Jetson prend en charge à la fois HDMI et DisplayPort pour la sortie écran. Nous supposerons qu'un câble Ethernet est également connecté dans les étapes suivantes.

Pas pour les maladroits !

Le support microSD du module Jetson est basé sur le principe du « form-fit ». Il est inséré et retiré en l'enfonçant - si vous utilisez une pince électronique et un tournevis fin, vous pouvez effectuer l'opération sans démonter la carte d'évaluation.

L'absence de caméra s'est avérée problématique lors de l'installation. Au lieu d'un capteur CCD, NVIDIA utilise deux connecteurs ordinaires des Raspberry Pi 4 et cartes similaires. Il est possible de connecter certaines caméras Raspberry Pi directement au Jetson. Dans les étapes suivantes, l'auteur utilise une caméra Raspberry Pi 2 connectée au port CAM0 via un câble FPC conçu pour le Raspberry Pi. Il faut noter que la version du câble destinée au Raspberry Pi 5 n'est pas compatible avec le Jetson. Différents modèles 3D pour connecter la caméra sont disponibles chez ThingiVerse, et leur impression simplifiera la tâche aux développeurs.

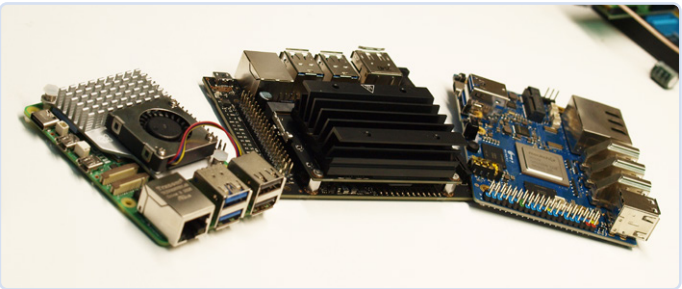


Figure 2. Le Jetson à côté d'un Raspberry Pi 5 et d'un OrangePi 5 Plus.

Figure 3. Le desserrage de deux vis permet de démonter le kit de développement.

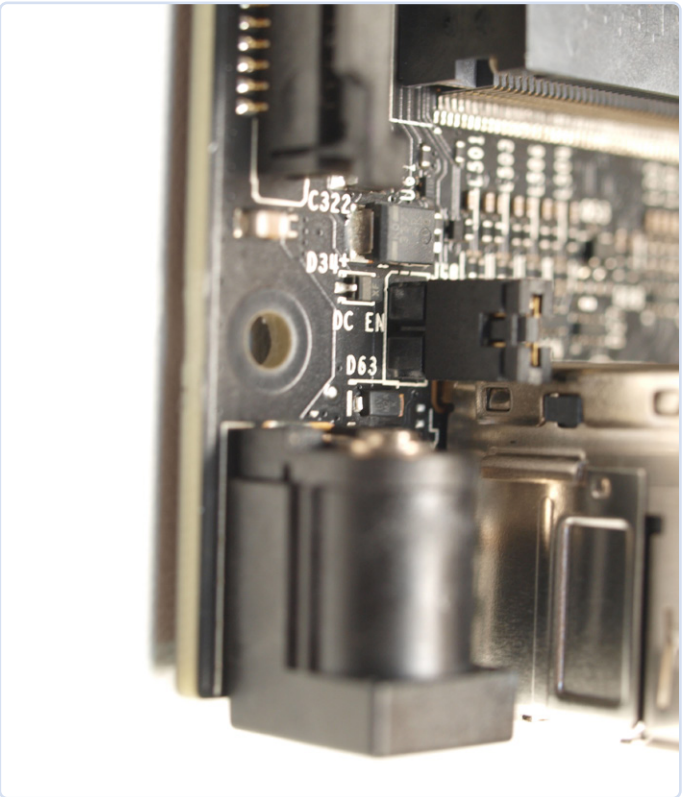


Figure 4. Ce cavalier est crucial.

Si vous souhaitez alimenter le NVIDIA Jetson avec l'alimentation MeanWell susmentionné, vous devez éviter un petit piège de débutant. Le cavalier montré branché dans la **figure 4** n'est pas branché à la livraison : dans ce cas, le connecteur DC n'est alors pas relié à l'alimentation. Lorsque l'alimentation est connectée, la LED d'état ne s'allume pas, ce qui peut prêter à confusion.
Lors du premier démarrage, l'ordinateur allume immédiatement l'écran HDMI - la reconfiguration de la carte microSD et d'autres initiations prennent un certain temps. Le système présente ensuite un bureau Ubuntu et vous demande de suivre l'assistant de bienvenue.
En général, il s'agit de l'assistant d'installation Ubuntu habituel, mais NVIDIA a ajouté une étape pour approuver les licences logicielles internes, entre autres choses. L'assistant de sélection du profil d'alimentation est important : il est recommandé de garder l'option par défaut. Elle garantit que le Jetson reçoit la puissance maximale.
Une fois cette étape terminée, il y en aura d'autres avant que la carte Jetson ne soit prête à l'emploi - des invites peuvent également apparaître sur le bureau pour redémarrer le système.

Premiers essais

NVIDIA s'appuie sur un système Ubuntu 18.04 plus ou moins compatible avec le Jetson. L'auteur aime configurer de tels processeurs pour l'accès à distance afin d'épargner la peine de passer du clavier du PC à celui de l'ordinateur de traitement.
Cependant, l'application *Settings* d'Ubuntu n'est pas adaptée à cette tâche, c'est pourquoi nous exécutons `sudo apt-get update` et `sudo apt-get upgrade` pour mettre à jour l'inventaire de paquets lors de la première étape. Toute requête doit être validée en appuyant sur *Enter* ; vous devez autoriser le redémarrage du système Docker.
Vous devez ensuite redémarrer le Jetson. En entrant `sudo apt install vino`, vous vous assurez que le serveur VNC est prêt à être utilisé.
Enfin, les commandes ci-dessous sont nécessaires pour que la configuration soit utilisable. Bien entendu, vous devez remplacer la chaîne de caractères `thepassword` par un mot de passe adapté.

```
tamhan@tamhan-desktop:~$ mkdir -p ~/.config/autostart
tamhan@tamhan-desktop:~$ cp /usr/share/applications/
vino-server.desktop ~/.config/autostart
tamhan@tamhan-desktop:~$ gsettings set org.gnome.Vino
prompt-enabled false
tamhan@tamhan-desktop:~$ gsettings set org.gnome.Vino
require-encryption false
tamhan@tamhan-desktop:~$ gsettings set org.gnome.Vino
authentication-methods "['vnc']"
tamhan@tamhan-desktop:~$ gsettings set org.gnome.Vino
vnc-password $(echo -n 'thepassword'|base64)
```

Après le redémarrage suivant, il est possible d'accéder au Jetson à l'aide du client VNC Reminna si un utilisateur est connecté. Notez toutefois que l'applet d'accès à distance dans l'application *Settings* ne fonctionne toujours pas.
Ensuite, vous pouvez effectuer un test initial de la connexion de la caméra en entrant la commande suivante :

```
tamhan@tamhan-desktop:~$ gst-launch-1.0 nvarguscamerasrc
! nvoverlaysink
```

L'aperçu de la caméra qui peut être activé avec cette commande de base n'apparaît que sur un moniteur connecté à la carte Jetson - le système communiquant avec l'ordinateur via VNC voit plutôt la sortie du terminal. Pour mettre fin au processus, il suffit d'envoyer un événement d'interruption en appuyant sur *Control* + *C*.
L'accès à la caméra est alors - en général - assuré par des méthodes connues du PC ou de la station de travail. Le fichier [5], qui montre la configuration d'un pipeline basé sur Open CV, est particulièrement intéressant.
Les commandes suivantes sont nécessaires pour le rendre exécutable localement :

```
tamhan@tamhan-desktop:~$ git clone https://github.com/
JetsonHacksNano/CSI-Camera
tamhan@tamhan-desktop:~$ cd CSI-Camera/
tamhan@tamhan-desktop:~/CSI-Camera$ python3 simple_
camera.py
```

La fenêtre de la caméra ouverte est alors également visible via VNC, car les informations ne sont pas envoyées directement au framebuffer du GPU Tegra.
Si la fenêtre de la caméra est à l'envers, vous pouvez modifier le paramètre `flip_method` dans le fichier :

```
def show_camera():
    window_title = "CSI Camera"
    print(gstreamer_pipeline(flip_method=2))
     video_capture = cv2.VideoCapture(gstreamer_
            pipeline(flip_method=2), cv2.CAP_GSTREAMER)
    if video_capture.isOpened():
    . . .
```

Interagir avec les broches d'E/S (via Python)

Les systèmes d'IA nécessitent des compétences complètement différentes de la part du développeur, qui ne correspond généralement que très peu au développement classique des systèmes embarqués. Dans la pratique, les personnes extérieures au domaine, ayant une formation en informatique, maîtrisent souvent mieux l'IA que les développeurs de systèmes embarqués. La partenaire de l'auteur, qui programme très efficacement en Java, résout les tâches d'IA plus rapidement - mais elle n'a que peu de connaissances en langage assembleur.
Le but de ce bref aperçu de l'architecture logicielle est de souligner que nous décrivons intentionnellement l'accès aux GPIO sur le Jetson en Python dans les étapes suivantes. La raison en est que la plupart des développements de systèmes d'intelligence artificielle orientés vers l'utilisateur sont faites avec Python : si l'on s'appuie ici sur le langage C, on se retrouve avec une interface native.
La distribution standard de Python n'inclut pas la gestion de paquets sur le Jetson, c'est pourquoi vous devez entrer les commandes

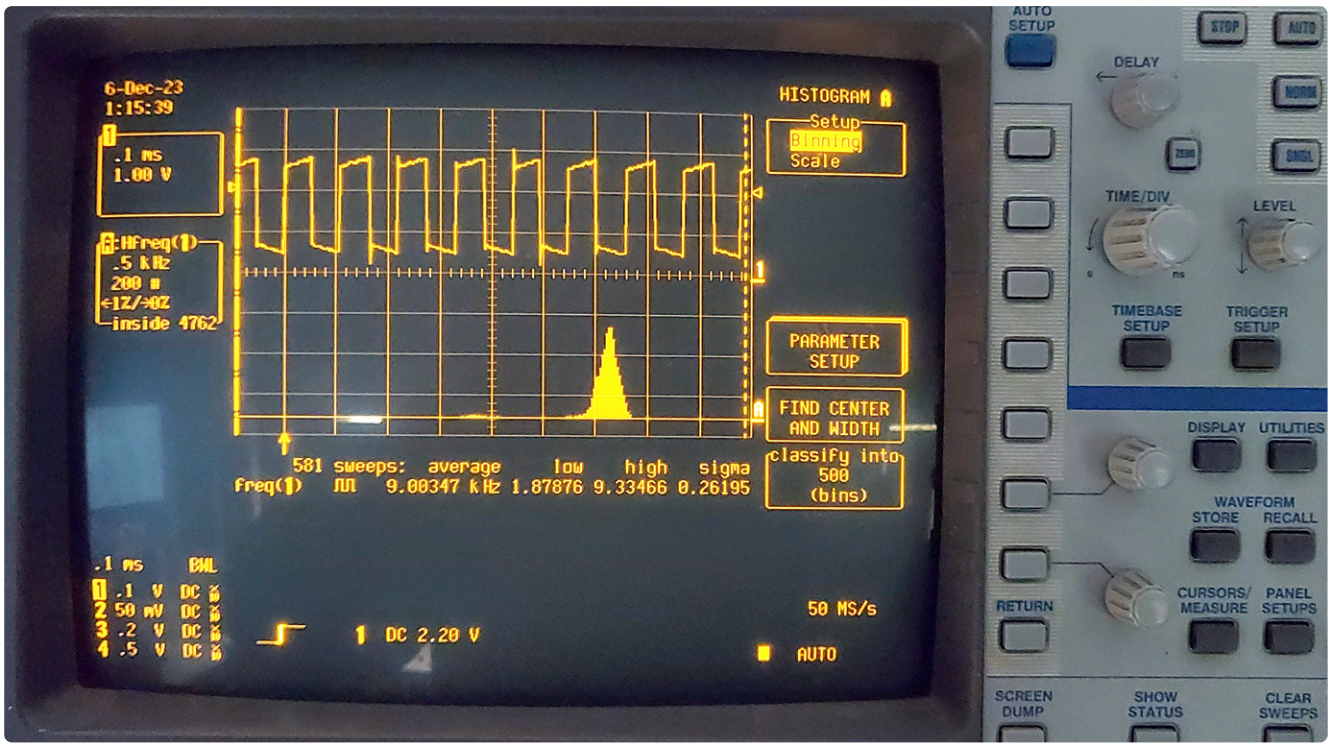


Figure 5. Le Jetson de NVIDIA produit des ondes carrées avec Python.

`sudo apt install python-pip` et `sudo apt install python3-pip`. L'étape suivante consiste à vérifier que les modules GPIO ont été correctement installés. Deux commandes sont également nécessaires ici, car les environnements Python ne sont (bien entendu) pas vraiment capables de partager leur référentiel de bibliothèques :

```
tamhan@tamhan-desktop:~$ sudo pip install Jetson.GPIO
tamhan@tamhan-desktop:~$ sudo pip3 install Jetson.GPIO
```

Nous ne pouvons pas présenter l'API GPIO Jetson dans son intégralité dans la limite de cet article. Sous [6], vous trouverez des exemples prêts à l'emploi qui expliquent l'API.
Le programme suivant est suffisant pour notre test :

```
import RPi.GPIO as GPIO
import time

output_pin = 18  # BCM pin 18, BOARD pin 12

def main():
    GPIO.setmode(GPIO.BCM)
# BCM pin-numbering scheme from Raspberry Pi
    GPIO.setup(output_pin, GPIO.OUT, initial=GPIO.HIGH)

    print("Starting demo now! Press CTRL+C to exit")
    try:
        while True:
            GPIO.output(output_pin,  GPIO.HIGH)
            GPIO.output(output_pin,  GPIO.LOW)
            GPIO.output(output_pin,  GPIO.HIGH)
            GPIO.output(output_pin,  GPIO.LOW)

    finally:
        GPIO.cleanup()

if __name__ == '__main__':
    main()
```

Ce qui est particulièrement intéressant ici, c'est que NVIDIA propose au développeur Jetson débutant plusieurs façons d'adresser les broches - nous utilisons ici la fonction `GPIO.BCM`, qui est basée sur le brochage du Raspberry Pi. La récompense de nos efforts est la capture d'écran présentée dans la **figure 5** - la stabilité de la fréquence n'est peut-être pas élevée, mais elle est plus que suffisante pour exécuter des fonctions IdO.
Il convient également de noter que l'exécution d'exemples de programmation de GPIO sans droits de superutilisateur peut parfois poser des problèmes. Voir [7] pour plus de détails à ce sujet.

Essais avec la fonction ML

En théorie, la disponibilité d'un système d'exploitation Linux complet et d'une interface USB3 (très rapide) facilite la réalisation de tests. Un exemple possible serait le modèle Stable Diffusion : en pratique, il est possible d'utiliser le Jetson comme générateur d'images à l'aide d'un runner (modifié).
Cependant, cette approche n'est pas recommandée dans la pratique

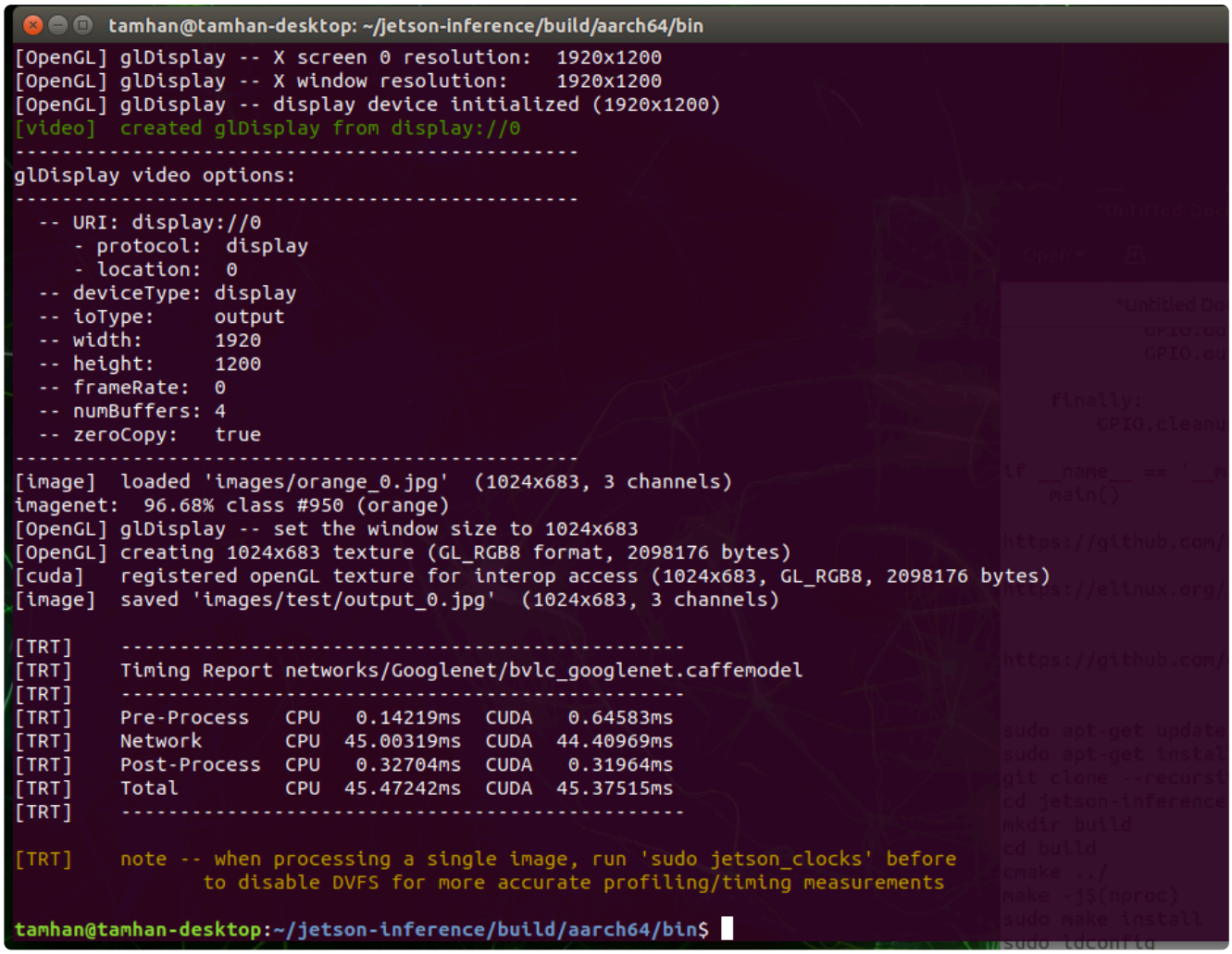


Figure 6. Le programme de démo fourni avec le Jetson n'est pas très « communicatif » au niveau de la ligne de commande.

: tant la mémoire graphique relativement faible de seulement 4 GB VRAM que le faible nombre de cœurs 128 (seulement) rendent la génération d'images lente. Sur Reddit, on trouve des rapports d'expérimentateurs de ML qui estiment jusqu'à 5 heures de calcul par image avec une taille de cluster de 512 × 512 pixels.

Un coup de pouce pour les frameworks les plus répandus

NVIDIA s'efforce de faciliter autant que possible l'implémentation de divers frameworks ML largement utilisés par les développeurs. Une liste des produits pris en charge, y compris des instructions d'installation optimisées, est disponible à l'adresse suivante : [8].

Il en va de même pour les solutions « de bout en bout » qui sont souvent présentées - l'entraînement d'un modèle nécessite tellement de puissance de calcul et de ressources que NVIDIA recommande de l'externaliser sur un ordinateur de bureau ou un ordinateur central dans la plupart des tutoriels - le Jetson est ensuite configuré avec les tailles des modèles prêt à l'emploi.
Sur [9], vous trouverez des exemples plus ou moins prêts à l'emploi qui illustrent les performances du Jetson de manière simple.
Pour utiliser ce modèle « powershow », vous devez télécharger et implémenter un logiciel NVIDIA. En théorie, l'utilisation d'un conteneur Docker est également possible ici - pour ceux qui ne sont pas familiers avec la technologie des conteneurs, la compilation locale est une alternative, qui peut être réalisée en entrant les commandes suivantes :

```
sudo apt-get update
sudo apt-get install git cmake libpython3-dev
python3-numpy
git clone --recursive --depth=1 https://github.com/
dusty-nv/jetson-inference
cd jetson-inference
mkdir build
cd build
cmake ../
make -j$(nproc)
sudo make install
sudo ldconfig
```

N'hésitez pas à répondre (par oui ou non) à la question concernant l'installation des composants d'entrainement - l'auteur a répondu Non dans les étapes suivantes pour économiser de la mémoire sur sa carte microSD, dont la taille n'est que de 32 GB.
Une fois la compilation terminée avec succès, vous trouverez une structure de projet relativement complexe dans le répertoire `~/jetson-inference/build/aarch64$/bin`, qui contient des

fichiers Python ainsi que divers fichiers binaires. Il est intéressant de noter que NVIDIA y a même placé des exemples de test prêts à l'emploi.
Tout d'abord, nous voulons utiliser le classificateur - il analyse les fichiers d'images et détermine ce qu'il faut voir dans l'image fournie :

```
tamhan@tamhan-desktop:~/jetson-inference/build/aarch64/
bin$ ./imagenet.py images/orange_0.jpg images/test/
output_0.jpg
```

L'exécution de cette commande pour la première fois prend un peu plus de temps car les modules fournis sont optimisés pour les besoins du système Jetson. Ensuite nous voyons les infos de timing dans la **figure 6**.
Pour visualiser les résultats du processus de ML, visualisez le fichier image généré - la commande suivante ouvre le dossier de sortie directement dans le gestionnaire de fichiers Nautilus :

```
tamhan@tamhan-desktop:~/jetson-inference/build/aarch64/
bin$ nautilus images/test/output_0.jpg
```

La récompense de vos efforts est l'image d'écran montrée dans la **figure 7**.

Analyse du fichier Python

Jetons maintenant un coup d'œil rapide à l'exemple de programme que nous venons d'utiliser. Le code source est accessible en entrant la ligne de commande suivante :

```
tamhan@tamhan-desktop:~/jetson-inference/build/aarch64/
bin$ gedit imagenet.py
```

Même à première vue, il est clair que la bibliothèque utilisée ici est liée au classique *ImageNet* - NVIDIA regroupe plusieurs systèmes d'IA couramment utilisés avec le kit de démarrage Jetson.
Le cœur de l'exemple de programme est une boucle infinie qui

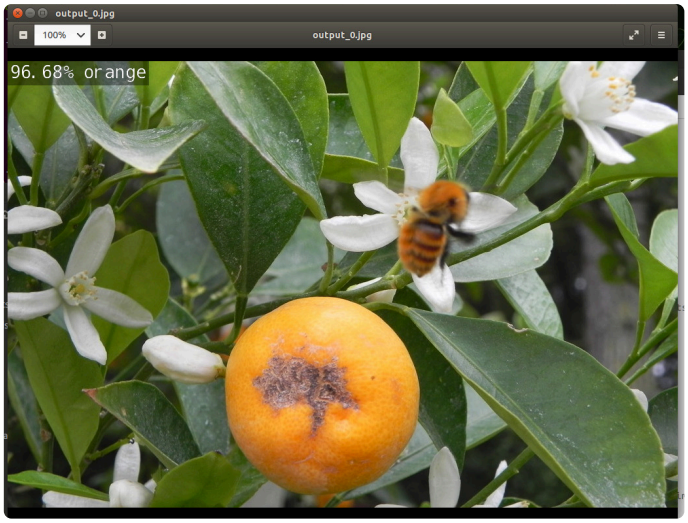


Figure 7. L'exemple de programme de NVIDIA s'est avéré être un détecteur de fruits exemplaire.

reçoit une image du flux d'entrée, la transmet au modèle de ML et affiche enfin les infos « calculées » (**listage 1**).
Pour cela, l'initialisation des flux de données et du modèle à utiliser est effectuée comme suit :

```
net = imageNet(args.network, sys.argv)
input = videoSource(args.input, argv=sys.argv)
output = videoOutput(args.output, argv=sys.argv)
font = cudaFont()
```

Un autre aspect intéressant est d'obtenir ou de compléter le paramètre `network`, qui fournit le nom du modèle à traiter. Ici, NVIDIA s'appuie sur la classe `ArgParser`, qui - normalement - se spécialise dans le traitement des paramètres fournis via la ligne de commande.
Dans le cas de cette déclaration, une valeur par défaut est introduite qui active et charge GoogLeNet :

```
parser.add_argument("--network",  type=str,
default="googlenet", help="pre-trained model to load
(see below for options)")
```

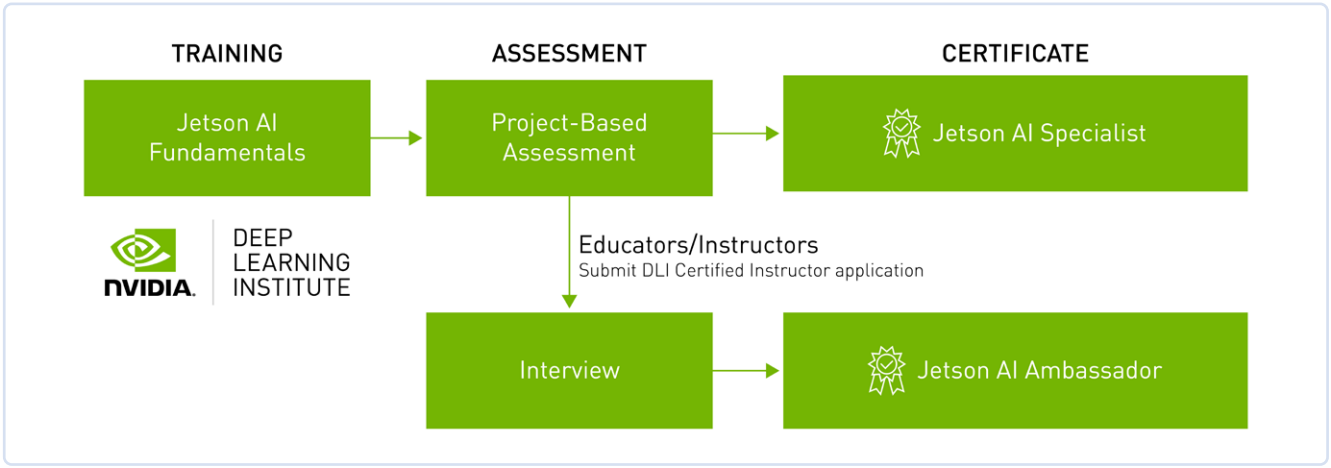


Figure 8. Apprentissage à deux voies de NVIDIA (source : NVIDIA [10])

Listage 1. Classification.

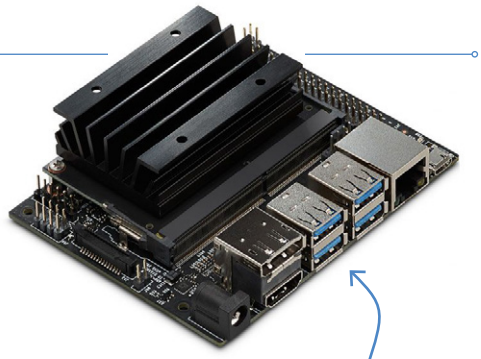
```
while True:
    img = input.Capture()

    if img is None: # timeout
        continue
        predictions = net.Classify(img, topK=args.topK)
        for n, (classID, confidence) in enumerate(predictions):
            classLabel = net.GetClassLabel(classID)
            confidence *= 100.0

            print(f"imagenet: % class # ()")

            font.OverlayText(img, text=f"% ",
                             x=5, y=5 + n * (font.GetSize() + 5),
                             color=font.White, background=font.Gray40)

    output.Render(img)
```



Produits

- **NVIDIA Jetson Nano Developer Kit (B01)**
  www.elektor.fr/19001

Note : formation (guidée) en ligne avec une certification

Le succès de l'ancienne Union soviétique dans plusieurs pays arabes et africains peut être en partie attribué au partenariat étroit en matière de formation - ce que le cadet apprend est ce qu'il utilise plus tard dans son travail. NVIDIA est clairement conscient de cette situation, c'est pourquoi la certification Jetson AI - comme le montre la **figure 8** - consiste en un cours de ML entièrement gratuit en deux parties. Il convient également de souligner que Nvidia offre un certificat récompensant l'achèvement de ce cours. Si vous êtes intéressé, nous vous encourageons à visiter le site [10]. Vous y trouverez de plus amples informations sur la manière d'organiser au mieux votre participation à la formation NVIDIA.

Avantages de Linux

Avec le Jetson, NVIDIA lance un hybride dans cette lutte entre tous les acteurs du domaine de l'IA. D'une part, les microcontrôleurs basse consommation dédiés, tels que le Maxim MAX78000, consomment beaucoup moins d'énergie. D'autre part, ces contrôleurs ne prennent pas en charge CUDA : un modèle pouvant fonctionner sur un PC ou un ordinateur central doit donc être modifié avant de pouvoir être utilisé avec ces puces dans des applications IdO.

En revanche, le NVIDIA Jetson ne constitue pas un GPU à part entière : tant en termes de consommation d'énergie que des versions CUDA prises en charge (CUDA 11 ne fonctionne pas), le module n'est pas un substitut à part entière d'un RTX 4000.

En fin de compte, l'implémentation est réussie si un modèle qui fonctionne sans problème sur un ordinateur ou une unité centrale doit être utilisé avec peu d'efforts et s'il dispose de suffisamment de ressources fournies par le Jetson. La disponibilité d'un système d'exploitation Linux signifie que relativement peu de travail de conversion est nécessaire pour un scientifique des données - se familiariser avec les API intégrées utilisées par Maxim et consorts demande beaucoup plus d'efforts. Le coût plus élevé du matériel peut donc être amorti rapidement et efficacement, en particulier pour les petites séries.

230740-04

Questions ou commentaires ?

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

LIENS

[1] Aperçu du portefeuille : https://www.nvidia.com/en-eu/autonomous-machines/embedded-systems/
[2] Liste de divers produits tiers : https://developer.nvidia.com/embedded/ecosystem
[3] Meilleur prix d'OEMSecrets : https://www.oemsecrets.com/compare/%20945-13450-0000-100
[4] Téléchargement du fichier image : https://developer.nvidia.com/jetson-nano-sd-card-image
[5] Fichier du pipeline basé sur un Open CV : https://github.com/JetsonHacksNano/CSI-Camera/blob/master/simple_camera.py
[6] Exemples prêts à l'emploi : https://github.com/NVIDIA/jetson-gpio/tree/master/samples
[7] Exécution d'exemples de programmes GPIO sans droits de superutilisateur : https://github.com/NVIDIA/jetson-gpio/issues/20
[8] Liste des produits pris en charge : https://elinux.org/Jetson_Zoo
[9] Exemples prêts à l'emploi : https://github.com/dusty-nv/jetson-inference
[10] Cours et certificats Jetson AI : https://developer.nvidia.com/embedded/learn/jetson-ai-certification-programs