MACHINE LEARNING SUR DES OBJETS CONNECTÉS AVEC TENSORFLOW LITE POUR L'AGRICULTURE VERTICALE

ALEXIS DUQUE
[Directeur Recherche et Développement de Rtone]

MOTS-CLÉS : TENSORFLOW LITE, MACHINE LEARNING, AGRICULTURE VERTICALE, EDGE COMPUTING

Tout au long de cet article, nous verrons comment déployer des algorithmes de Deep Learning sur des objets connectés grâce à TensorFlow Lite. Nous verrons comment l'utiliser pour concevoir une « ferme verticale » capable de prédire et optimiser la production de légumes, aussi bien chez soi que dans des pays en voie de développement où la connexion internet est intermittente.

Aujourd'hui, les utilisateurs d'objets connectés demandent de plus en plus de fonctionnalités et s'attendent souvent à une interaction presque humaine avec les appareils qu'ils utilisent. Ces objets doivent donc être de plus en plus « intelligents » et le développement d'une intelligence artificielle devient alors un attribut clé dans leur conception. Celle-ci repose sur des algorithmes de *Machine Learning*, comme l'apprentissage profond, en anglais *Deep Learning*, souvent intégrés à des applications hébergées dans le *Cloud*.

Grâce à l'augmentation des performances des microcontrôleurs et l'intérêt grandissant pour le paradigme de l'« Edge Computing », plusieurs bibliothèques de *Machine Learning* destinées aux plateformes contraintes, telles que les smartphones, les box domotiques, voire les microcontrôleurs ont été développées. Leur objectif est de déporter l'intelligence et le traitement à la périphérie du réseau. Utiliser du *Machine Learning* « à la périphérie » présente en effet plusieurs avantages comme la réduction de la latence, le respect de la vie privée, et un fonctionnement sans connexion internet. Citons comme exemples **Snips [1]** ou **DeepSpeech** de **Mozilla [2]**, tous les deux destinés à la reconnaissance vocale, mais également **TensorFlow Lite [3]**, plus généraliste, dont nous verrons l'utilisation en détail dans cet article.

En fait, alors que le développement et plus particulièrement l'entraînement d'un modèle de *Deep Learning* nécessitent une capacité de calcul importante, l'inférence, c'est-à-dire l'exécution d'un modèle préalablement entraîné pour faire une prédiction, nécessite moins de ressources et peut être exécutée en périphérie. Cela permet aux développeurs d'ajouter de l'intelligence aux objets connectés pour effectuer des tâches telles que la détection d'anomalies, la reconnaissance vocale ou des prédictions, sans dépendre d'un serveur *Cloud* et donc d'une connexion à Internet.

Cet article se concentrera sur l'utilisation de TensorFlow Lite sur les objets connectés et plus précisément pour le développement d'une « ferme verticale ». L'apprentissage machine et l'intelligence artificielle nous aideront à optimiser le rendement des légumes et à prévoir la date de récolte des plantes.

La première section présentera brièvement le concept d'agriculture verticale, avant d'introduire TensorFlow Lite et ses différences avec **TensorFlow [4]**. Ensuite, je présenterai la ferme verticale connectée que nous avons développée et pourquoi nous utilisons TensorFlow Lite. La partie principale sera un tutoriel sur la conception d'un modèle de prédiction de croissance de laitues à l'aide de TensorFlow, son déploiement et son utilisation avec Tensorflow Lite sur un **Raspberry Pi** pour prédire la date de récolte de la salade.

Enfin, je conclurai cet article en présentant les performances du système et un retour d'expérience issu de ce projet.

1. QU'EST-CE QUE L'AGRICULTURE VERTICALE ?

L'agriculture verticale est une nouvelle méthode de culture en « strates » superposées verticalement, dans un environnement climatique entièrement contrôlé et monitoré. L'agriculture verticale vise à minimiser l'utilisation de l'eau et à maximiser la productivité en faisant pousser des cultures, sans sol, et qui nécessitent de petites quantités d'eau riche en nutriments.

Ainsi, l'agriculture verticale pourrait contribuer à répondre de manière durable à la demande alimentaire mondiale croissante en réduisant les chaînes de distribution, en produisant des plantes et des légumes frais à proximité des populations pour offrir des émissions de polluants plus faibles, en fournissant des produits plus nutritifs, avec un meilleur goût et une meilleure apparence.

Au cours des cinq dernières années, de nombreuses entreprises ont commencé à développer des systèmes d'agriculture verticale en choisissant différentes structures comme des conteneurs d'expédition, des bâtiments, des tunnels ou des armoires comme le système que je présenterai dans cet article.

Deux catégories de techniques d'agriculture sans sol sont utilisées dans l'agriculture verticale : les systèmes reposants sur l'hydroponie et ceux utilisant l'aéroponie.

Dans les systèmes hydroponiques, les racines des légumes verts sont submergées dans un courant très peu profond de solutions aqueuses contenant des macronutriments (azote, phosphore, potassium, etc.). Cette technique est appelée *Nutrient Film Technique* (NFT). Facultativement, des milieux inertes tels que le gravier ou le sable peuvent être utilisés comme substituts de sol pour soutenir les racines. Cette technique est appelée *Ebb & Flow*.

Contrairement à la culture hydroponique, l'aéroponie ne nécessite aucun milieu liquide ou solide pour la croissance des plantes. Au lieu de cela, une solution liquide contenant des nutriments est diffusée avec un jet à haute pression, on parle alors d'*High Pressure Aeroponics* (HPA) ; avec un brouillard, on parle alors de nébulisation. La nébulisation aéroponique est la technique de culture sans sol la plus durable, car elle utilise jusqu'à 90 % de moins d'eau que les systèmes hydroponiques conventionnels les plus efficaces et ne nécessite pas le remplacement du milieu de culture.

Au-delà des considérations techniques et écologiques, le système d'irrigation a un fort impact sur la croissance des légumes. C'est pourquoi c'est un paramètre clé à prendre en compte dans la ferme verticale et le développement de l'intelligence artificielle que je vais introduire dans la prochaine section.

2. PRÉSENTATION DE TENSORFLOW ET TENSORFLOW LITE

TensorFlow est une plateforme logicielle qui a été développée par l'équipe **Google Brain**, initialement pour une utilisation interne à Google. Il s'agit d'une bibliothèque de logiciels libres et *open source* publiée sous la licence **Apache 2.0** à la fin de 2015. Elle permet d'effectuer un large éventail de tâches de *Machine Learning*, telles que la conception et l'entraînement de réseaux de neurones. Depuis janvier 2020, la dernière version officielle est TensorFlow 2.1.

TensorFlow fournit des API **Python** (pour la version 3.7) et **C** stables. Cependant, d'autres langages de programmation sont pris en charge, mais sans garantie de rétrocompatibilité des API : **C++**, **Go**, **Java**, **JavaScript** et **Swift**. Dans cet article, nous utiliserons l'API Python.

La plateforme TensorFlow est particulièrement complexe et inclut de nombreux composants, comme **Tensorboard**, pour déboguer et étudier les étapes d'apprentissage d'un modèle ; **TensorFlow.js**, pour exécuter un modèle et faire des prédictions dans un navigateur web ou TensorFlow Lite pour les appareils à plus faibles capacités de calcul, comme les smartphones.

TensorFlow Lite fournit tous les éléments de base pour convertir et optimiser un modèle construit et entraîné avec TensorFlow, et les exécuter ensuite sur des appareils mobiles et objets connectés. Cependant, TensorFlow Lite, contrairement à TensorFlow, ne peut pas être utilisé pour développer et entraîner un modèle de *Machine Learning*.

TensorFlow Lite possède deux composants : le convertisseur TensorFlow Lite et le moteur d'exécution TensorFlow Lite. Le convertisseur TensorFlow Lite offre toutes les API et les optimiseurs nécessaires pour réduire l'empreinte mémoire d'un modèle TensorFlow, augmenter sa vitesse d'exécution et réduire les coûts de calcul. La section 3.3 montre plus en détail comment les utiliser.

Le principe habituel d'utilisation de Tensorflow et Tensorflow Lite pour concevoir un modèle de *Deep Learning* est le suivant :

- Le *data scientist*, sur son ordinateur, effectue des tâches d'analyse et de prétraitement des données avec des librairies comme **Pandas [5]** ou **Numpy [6]**, avant d'utiliser TensorFlow pour concevoir et entraîner un réseau de neurones, souvent dans un *notebook* **Jupyter [7]**. Cette phase d'entraînement peut éventuellement être déportée sur un serveur de calcul disposant d'un GPU.
- Ensuite, toujours sur son ordinateur, le chercheur utilise **TensorFlow Lite Converter**, qui est fourni avec la distribution TensorFlow, pour optimiser et convertir le modèle formé en un modèle `.tflite` optimisé pour les objets connectés. Ce modèle `.tflite` est ensuite déployé sur le périphérique, sous la forme d'une mise à jour logicielle, par exemple.
- Enfin, au moment de l'exécution, l'objet connecté ou le smartphone utilise le moteur d'exécution TensorFlow Lite Interpreter, pour exécuter le modèle et faire des prédictions en utilisant les données en direct. Un point essentiel ici est que la distribution TensorFlow n'est pas installée sur l'appareil, mais seulement le moteur d'exécution TensorFlow Lite, totalement autonome. Selon la plateforme de l'appareil périphérique, le moteur d'exécution TensorFlow Lite est fourni soit dans un paquet AAR distribué sur **JCenter** pour **Android**, un paquet Python **wheel** ou une bibliothèque partagée en C pour les appareils embarqués.

3. TENSORFLOW LITE POUR L'AGRICULTURE VERTICALE

3.1 Description de notre armoire agricole connectée

Nous avons conçu l'armoire pour l'agriculture verticale d'intérieur illustrée à la figure 1. L'armoire comporte 4 étagères pouvant contenir chacune 12 plantes réparties en grille.

Fig. 1 : La ferme connectée que nous avons développée. Nous voyons ici les différentes étagères, ainsi que plusieurs plants de laitue à différents stades de croissance.

Le système complet de ferme verticale est divisé en 4 composants, comme le résume la figure 2 :

1. l'armoire, les capteurs, les actionneurs et les systèmes d'irrigation ;
2. la carte mère, qui est ici un Raspberry PI 3B+ ;
3. le Cloud ;
4. l'ordinateur de l'équipe R&D.

L'armoire possède 2 caméras par étage ainsi que 8 capteurs pour mesurer les paramètres suivants : le CO_2 (en ppm), l'oxygène dissous (en ppm), la conductivité électrique (en µS/cm), le potentiel d'oxydoréduction (en mV), le PPFD (*Photosynthetic Photon Flux Density*, en µmol/m²/s), le pH de l'eau, l'humidité (en %) et la température (en °C). Notre prototype met en œuvre les 4 différents systèmes d'irrigation présentés dans la section 1, un sur chaque étage, afin d'étudier leur impact sur la croissance et la qualité des plantes. De plus, des actionneurs permettent de contrôler la température, le niveau de CO_2 et la luminosité.

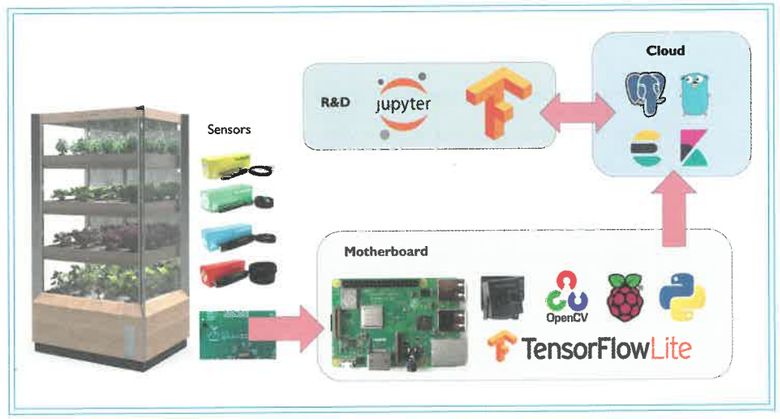

Fig. 2 : Schéma du système connecté pour l'agriculture verticale que nous avons développé.

Le Raspberry Pi est le cerveau du système. Il collecte les données des capteurs, contrôle les systèmes d'irrigation et les actionneurs. Il calcule également la surface foliaire des plantes en utilisant les caméras et plusieurs algorithmes de traitement d'images développés avec **OpenCV [8]**. La figure 3 illustre ce processus de calcul de la surface foliaire. En parallèle, il exécute en temps réel l'algorithme d'apprentissage profond de prédiction de la croissance des plantes grâce au moteur d'exécution TensorFlow Lite. Toutes les données sont « mises en cache » localement sur le Raspberry Pi, avant d'être transférées, lorsqu'Internet est disponible, vers l'API de notre serveur *Cloud*. Toutes ces fonctions ont été développées en Python.

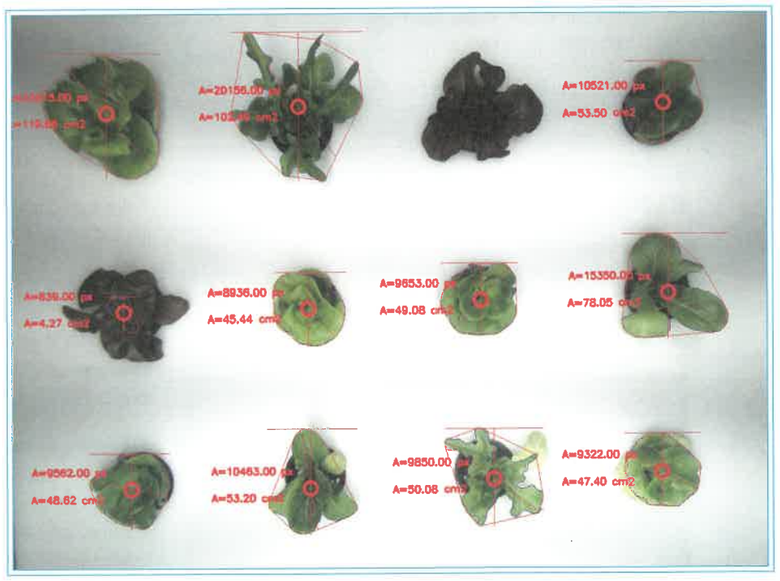

Fig. 3 : Exemple de calcul de la surface foliaire de plants de laitues avec OpenCV.

Le *Cloud* possède 3 fonctions principales :

1. une API développée en **Go** pour collecter les données de la ferme verticale (envoyée par le Raspberry Pi) ;
2. une base de données **PostgreSQL** pour stocker les données, ainsi que les informations détaillant les conditions expérimentales, climatiques et les caractéristiques des plantes (la variété, la date de plantation, etc.) principalement à des fins de recherche agronomique ;
3. une interface web pour le monitoring et la visualisation des données, développée avec la pile **ELK** (**Elasticsearch / Logstash / Kibana**) **[9]**.

3.2 Construire un modèle de prédiction du poids de la laitue basé sur le réseau de neurones

Dans un problème de régression, nous visons à prédire la production d'une valeur continue, comme une température ou un poids. Un problème de classification quant à lui vise à déterminer une classe, ou catégorie, à partir d'une liste de catégories.

Pour construire notre modèle, nous utiliserons la nouvelle API `tf.keras` introduite dans TensorFlow 2.0.

Notre modèle nous permettra de prédire le poids « frais » de la laitue, à une certaine date après la plantation. Il s'agit donc d'un modèle de régression.

3.2.1 Configurer son environnement de R&D sur son ordinateur

Pour cela, nous utiliserons Python 3.7 avec un environnement virtuel et installerons la librairie TensorFlow en utilisant `pip` :

1. Vérification des différentes versions :

```
$ python3 --version
$ pip3 --version
$ virtualenv --version
```

2. Création de l'environnement virtuel :

```
$ virtualenv --system-site-packages -p
python3 ./venv
# activate the environment
$ source ./venv/bin/activate
```

3. Installation proprement dite :

```
$ pip install --upgrade pip
$ pip install --upgrade tensorflow=2.0
pandas numpy pathlib
```

Ensuite, vérifiez que votre installation est correcte :

```
$ python -c "import tensorflow as
tf;print(tf.reduce_sum(tf.random.
normal([1000, 1000])))"
```

3.2.2 Construire et entraîner un réseau de neurones

Pour simplifier et faciliter la compréhension, considérons les 10 caractéristiques suivantes : la surface foliaire, la somme cumulée (`cumsum`) de la lumière, la somme cumulée du CO_2, la somme cumulée de la température, le nombre de jours après la plantation (`DaP`), le poids et le type de système d'irrigation que l'on prendra soin d'encoder en *One-Hot* avec 4 colonnes : chaque classe devient une colonne à part entière qui prend ses valeurs dans `{0,1}`, indiquant si l'observation utilise ce système d'irrigation ou non.

Notez que nous travaillons sur des données réelles provenant de capteurs qui peuvent tomber en panne ou être mal calibrés. Pour éviter des résultats aberrants, il est donc indispensable de réaliser un prétraitement comme l'échantillonnage, le filtrage et la normalisation, avant qu'elles ne soient exploitées pour entraîner un réseau de neurones, ou même exécuter le modèle. Cette étape n'est pas traitée dans cet article.

Le tableau 1 ci-contre donne un échantillon de notre jeu de données.

Nous avons divisé notre ensemble de données en deux, respectivement pour l'entraînement et le test. Nous séparons ensuite les 9 caractéristiques (qui seront l'entrée du réseau de neurones) du poids, c'est-à-dire la valeur que nous voulons prédire, et qui sera la sortie du réseau de neurones.

Construisons maintenant notre modèle. Ici, nous utiliserons un modèle simple `Sequential` avec deux couches denses cachées, et une couche de sortie qui renvoie une valeur unique et continue : le poids de la laitue. Nous choisissons seulement 2 couches cachées, car il n'y a pas beaucoup de données d'entraînement et seulement 9 caractéristiques. Dans ce cas, un petit réseau avec peu de couches

Weight	DaP	CumCO2	CumLight	CumTemp	LeafArea	NFT	HPA	Ebb&Flood	Nebu
135.0	10	4	1420.0	8623.0	156	0.0	1.0	0.0	0.0

Tableau 1.

cachées est recommandé pour éviter le surapprentissage. Nous avons empiriquement choisi la fonction d'activation de l'unité linéaire rectifiée, `ReLU`, très souvent utilisée. Enfin, nous avons choisi la fonction de perte de l'erreur quadratique moyenne, *Mean Squared Error*, qui est utilisée pour les problèmes de régression (d'autres fonctions de perte sont utilisées pour les problèmes de classification).

```
import tensorflow as tf
...

model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation=tf.nn.relu,
input_shape=[len(train_dataset.keys())]),
    tf.keras.layers.Dense(64, activation=tf.nn.relu),
    tf.keras.layers.Dense(1)
])

optimizer = tf.keras.optimizers.RMSprop(0.001)
model.compile(loss='mean_squared_error',
    optimizer=optimizer,
    metrics=['mean_absolute_error', 'mean_squared_error'])
```

Le modèle est présenté à la figure 4. Notez que le nombre d'entrées de notre réseau de neurones est de 9, ce qui correspond au nombre de paramètres utilisés pour prédire le poids.

Nous entraînons alors le modèle en utilisant l'ensemble du *dataset* d'entraînement :

```
model.fit(train_features, train_weights, epochs=100,
validation_split=0.2, verbose=0)
```

Nous pouvons maintenant sauvegarder et exporter le modèle entraîné. Dans notre cas, sa taille est de 86077 octets :

```
model_export_dir= "./models/lg_weight/"
tf.saved_model.save(model, model_export_dir)
root_directory = pathlib.Path(model_export_dir)
tf_model_size = sum(f.stat().st_size for f in root_
directory.glob('**/*.pb') if f.is_file())
print("TF model is {} bytes".format(keras_model_size))
```

3.3 Convertir et optimiser le modèle TensorFlow entraîné en un modèle TensorFlow Lite

Voici maintenant la partie importante de ce tutoriel. En utilisant le modèle entraîné préalablement, nous allons convertir le modèle au format TensorFlow Lite avec les optimisations par défaut et le sauvegarder :

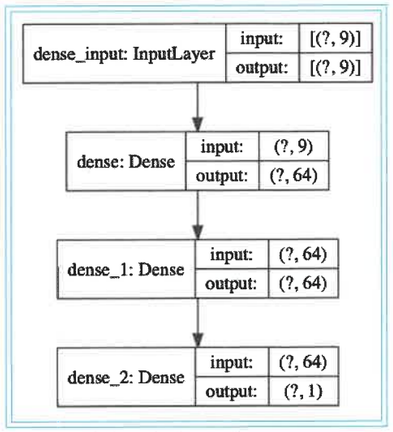

Fig. 4 : Schéma du réseau de neurones, comprenant une couche d'entrée, 2 couches denses et une couche de sortie.

```
converter = tf.lite.TFLiteConverter.
from_keras_model(model)
tflite_model = converter.convert()
open('./models/model.tflite', "wb").
write(tflite_model)
```

Si nous regardons la taille du modèle, nous constatons qu'elle est maintenant de `20912` octets. C'est 4 fois moins que le modèle original de TensorFlow :

```
model_size = os.path.getsize(
'./models/model.tflite')
print("TFLite model is {}
bytes".format(model_size))
```

Nous pouvons encore réduire la taille du modèle en utilisant les optimiseurs proposés par TensorFlow Lite Converter. La forme la plus simple d'optimisation post-entraînement quantifie les poids du réseau de neurones, initialement représentés sur 32 bits, sur 8 bits. C'est ce que l'on appelle également la quantification « hybride » :

```
converter.optimizations = [tf.
lite.Optimize.OPTIMIZE_FOR_SIZE]
tflite_quant_model = converter.
convert()
```

La taille du modèle n'est plus que de **8672** octets. C'est dix fois moins que le modèle TensorFlow classique.

3.4 Faire des prédictions au sein de la ferme verticale sur un Raspberry Pi

La section précédente a présenté les bases pour résoudre un problème de régression appliqué à la prédiction du poids de la laitue, et montré ensuite comment convertir et optimiser un modèle entraîné au format TensorFlow Lite.

Nous sommes maintenant prêts à déployer le modèle optimisé **model.tflite** sur notre ferme verticale et plus précisément sur le Raspberry Pi qui la contrôle.

N'oubliez pas que sur les objets connectés et smartphones, seul le moteur d'exécution TensorFlow Lite doit être installé.

Comme sur l'ordinateur de R&D, nous utiliserons Python 3.7 et **pip** pour installer le paquet *wheel*, qui ne contient cette fois que TensorFlow Lite Interpreter.

Dans notre cas, le système d'exploitation du Raspberry Pi est **Raspbian Buster**. Nous installons donc le paquet Python comme suit :

```
$ pip3 install https://
dl.google.com/coral/python/
tflite_runtime-2.1.0-cp37-
cp37m-linux_armv7l.whl
```

Les paquets destinés à d'autres systèmes d'exploitation et matériels sont disponibles sur le site web de Tensorflow Lite **[10]**.

Pour éviter des résultats aberrants, nous veillerons, comme lors de la conception du modèle, à réaliser un prétraitement des données. Ensuite, il est assez facile d'exécuter le modèle et de faire une prédiction en utilisant l'API Python.

La première fois, nous devons allouer la mémoire aux tenseurs :

```
import tflite_runtime.interpreter as tflite

interpreter = tflite.Interpreter(model_content=tflite_
model_ffile)
interpreter.allocate_tensors()
```

Nous copions ensuite les données des capteurs dans les tenseurs d'entrée, invoquons l'interpréteur et enfin obtenons la prédiction en venant lire les valeurs du tenseur de sortie. Dans l'extrait suivant, **input_tensor** est un tableau qui contient les caractéristiques d'entrée (c'est-à-dire le **cumsum** de lumière, le **cumsum** de température, etc.). Notez que **tensor_index** correspond au nombre de caractéristiques en entrée :

```
interpreter.set_tensor(tensor_index=9, value=input_tensor)
# run inference
interpreter.invoke()
# tensor_index is 0 since the output contains only a
single value
weight_inferred = interpreter.get_tensor(tensor_index=0)
```

4. RÉSULTATS ET PERFORMANCES

4.1 Précision

Le temps est venu d'évaluer la précision du modèle. Voyons dans quelle mesure le modèle se généralise en utilisant les données de test, que nous n'avons pas utilisées lors de l'entraînement du modèle. Cela nous indique dans quelle mesure nous pouvons nous attendre à ce que le modèle soit juste, quand nous l'utiliserons dans le monde réel.

Tout d'abord, pour évaluer en un coup d'œil la précision, nous pouvons tracer le graphique de la figure 5. Les croix bleues montrent les valeurs de poids prédites par le modèle en fonction des valeurs réelles pour l'ensemble des données de test. L'erreur peut ainsi être matérialisée comme la distance entre les croix bleues et la ligne orange.

Une métrique souvent utilisée pour évaluer les modèles de régression est l'erreur moyenne absolue en pourcentage, ou en anglais *Mean Absolute Percent Error* (MAPE), mesure l'écart entre les valeurs prévues et les valeurs observées.

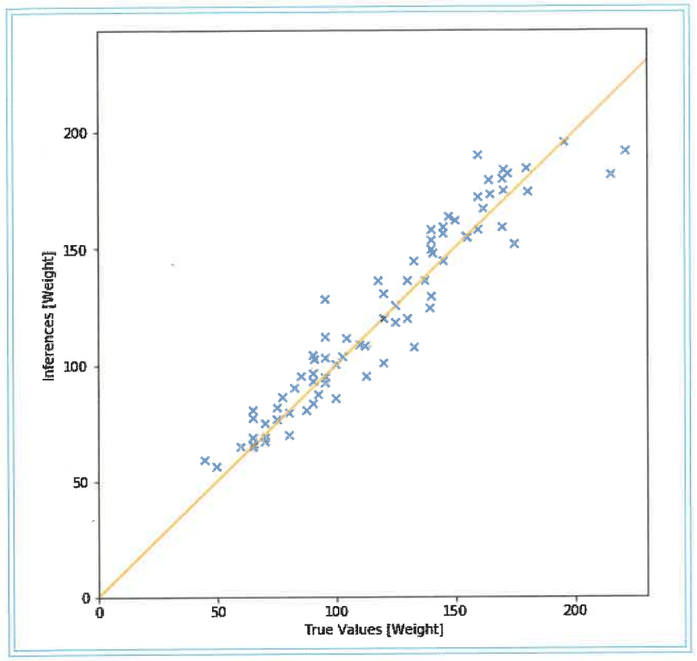

Fig. 5 : Le poids de laitue à différents stades de croissance, prédit par notre modèle en fonction des valeurs réelles en utilisant le jeu de données de test. La droite d'équation y = x en orange matérialise une prédiction parfaite.

Pour notre jeu de données de test, l'erreur moyenne absolue en pourcentage est de 9,44 %, ce qui est suffisamment précis pour les utilisateurs d'une ferme verticale.

4.2 Taille du modèle

En ce qui concerne la taille du modèle, nous avons vu juste avant qu'elle peut être réduite de **86077** octets à **8672** octets, en utilisant le convertisseur TensorFlow Lite avec quantification post-entraînement.

4.3 Délai d'exécution

Enfin, nous installons la distribution TensorFlow complète sur un Rasperry Pi de test et comparons le temps d'exécution de l'inférence entre les deux versions du modèle :

- notre modèle original, sans l'étape de conversion avec TensorFlow Lite et donc sans les optimisations, exécuté avec le moteur d'exécution de TensorFlow ;
- note modèle **.tflite** optimisé avec **TensorFlow Lite Converter** et exécuté avec le moteur d'exécution **TensorFlow Lite Interpreter** ;

Pour obtenir un résultat significatif, le Raspberry Pi n'exécute pas en parallèle notre logiciel de pilotage de la ferme verticale. Nous exécutons consécutivement 10 000 inférences et prenons la moyenne.

Les résultats sont rassemblés dans la figure 6 et montrent que l'inférence avec TensorFlow Lite avec un modèle optimisé est 3 à 4 plus rapide qu'avec TensorFlow sans optimisations. Elle est pour la version **.tflite** du modèle en moyenne de 6,2 ms contre 23,3 ms sans optimisations. Cette augmentation de la vitesse d'exécution est en grande partie due à l'opération de quantification, une simplification du graphe de notre réseau de neurones, et à l'utilisation au moment de l'inférence d'opérateurs optimisés pour la plateforme.

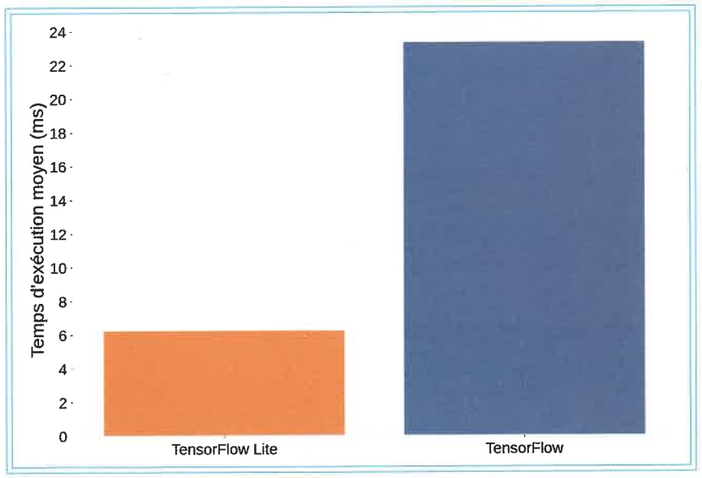

Fig. 6 : Le temps d'exécution moyen du modèle de prédiction du poids de la laitue, optimisé avec TensorFlow Lite (orange) et sans optimisations (bleu).

CONCLUSION

TensorFlow Lite nous permet de prévoir avec précision la date de récolte de la laitue qui pousse dans notre ferme verticale, sans avoir besoin d'un serveur puissant et avec une connexion intermittente au serveur *Cloud*. En réduisant le coût de calcul et la taille du modèle entraîné, l'inférence peut être exécutée sur un objet connecté, comme un Raspberry Pi, en utilisant l'interpréteur de TensorFlow Lite. Cette tâche a un faible coût et permet au Raspberry Pi de gérer toutes les autres tâches critiques qui font fonctionner notre ferme verticale. En outre, TensorFlow Lite peut également être utilisé sur des objets connectés aussi petits que les microcontrôleurs **ARM Cortex-M**. C'est notamment le cas de l'**Arduino Nano 33 BLE Sense [11]**.

TensorFlow Lite présente cependant plusieurs limites. Seuls certains opérateurs sont pris en charge, ce qui limite le type de modèle implémenté. Par exemple, les réseaux neuronaux récurrents ne sont pas entièrement pris en charge. Cependant, comme j'ai pu le constater, l'équipe de Google comble actuellement cette lacune et affirme vouloir prendre en charge tous les modèles dont dispose TensorFlow aujourd'hui. Une autre limite est l'absence de support pour l'apprentissage par renforcement qui nécessite plus de capacités de calcul, puisqu'un apprentissage incrémental est effectué sur l'appareil sur lequel il est déployé. Ce défi peut néanmoins être surmonté en utilisant un dispositif disposant d'un composant dédié, comme la carte **Google Coral [12][13]** qui dispose d'un TPU.

REMERCIEMENTS

Je tiens à remercier mes collègues au sein de **Rtone** qui ont participé à la réalisation de ce projet, Clément Michaud, Benjamin Vincent et Charly Hamy, ainsi que Bénédicte Pariaud, de la startup **La Grangette** et son fondateur Thibaut Pradier, de nous avoir fait confiance en nous confiant sa réalisation. Merci également à Mathilde Humbert pour sa précieuse relecture et ses remarques constructives. ■

POUR ALLER PLUS LOIN

Pour approfondir l'utilisation de TensorFlow, je recommande la lecture de la seconde édition du très bon livre d'Aurélien Géron, ainsi que l'ouvrage de Pete Warden et Daniel Situnayake consacré à Tensorflow Lite et le *Machine Learning* sur les objets connectés.

A. GÉRON, *« Hands-on Machine Learning With Scikit-Learn, Keras and TensorFlow »*, O'Reilly, 2019.

P. WARDEN et D. SITUNAYAKE, *« TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers »*, O'Reilly, 2019.

RÉFÉRENCES

[1] Site officiel de snips.io : https://snips.io

[2] DeepSpeech de Mozilla : https://hacks.mozilla.org/2019/12/deepspeech-0-6-mozillas-speech-to-text-engine/

[3] Site officiel de TensorFlow Lite : https://www.tensorflow.org/lite/

[4] Documentation de TensorFlow : https://www.tensorflow.org/

[5] Documentation de Pandas : https://pandas.pydata.org/

[6] Documentation de Numpy : https://numpy.org/

[7] Site officiel de Jupyter : https://jupyter.org/

[8] Documentation d'OpenCV : https://docs.opencv.org/4.2.0/

[9] Site officiel d'ELK : https://www.elastic.co/what-is/elk-stack

[10] Documentation Python de TensorFlow Lite : https://www.tensorflow.org/lite/guide/python

[11] Arduino Nano 33 BLE Sense : https://store.arduino.cc/arduino-nano-33-ble-sense

[12] Site officiel de Google Coral : https://coral.ai/

[13] T. COLOMBO, *« Utilisation d'un accélérateur matériel : test du TPU Coral USB Accelerator »*, *GNU/Linux Magazine n°106*, janvier 2020 : https://connect.ed-diamond.com/GNU-Linux-Magazine/GLMFHS-106/Utilisation-d-un-accelerateur-materiel-test-du-TPU-Coral-USB-Accelerator