Tensorflow Lite : Implémentations des opérateurs optimisés

Thaddée BRICOUT

CEA/DRT/LIST/DSCIN/LFIM / Grenoble

10 mai 2022

Sommaire

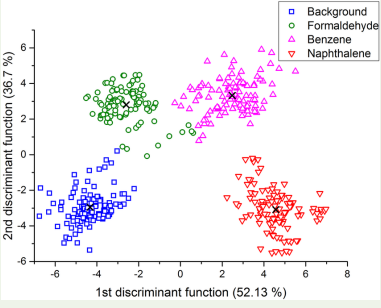
1. Rappel IA
   - Rappel IA : Exemple de classifieurs classiques
   - Rappel IA : Exemple : le réseau de neurones convolutifs
   - Rappel IA : Les réseaux de neurones, comment ça marche ?
   - Rappel IA : Utilisation d'un réseau de neurones
2. Tensorflow
   - Tensorflow : Qu'est-ce que c'est ?
   - Tensorflow : Création d'un réseau de neurones
3. Tensorflow Lite
   - Tensorflow lite : Pourquoi ce choix ?
   - Tensorflow lite : Organisation de la bibliothèque C++
   - Tensorflow lite : Exemple d'un opérateur générique : La convolution
   - Tensorflow lite : Fonctionnement de l'Inférence
4. Conclusion
   - Conclusion : Les étapes pour l'implémentation des opérateurs optimisés
   - Avez-vous des questions?

Rappel IA : Exemple de classifieurs classiques

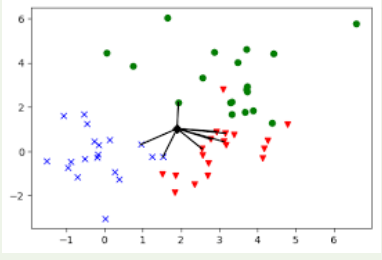
Examples de données d'entrées

- Pression sanguines
- Températures
- Taux d'oxygène dans le sang

Analyse discriminante linéaire

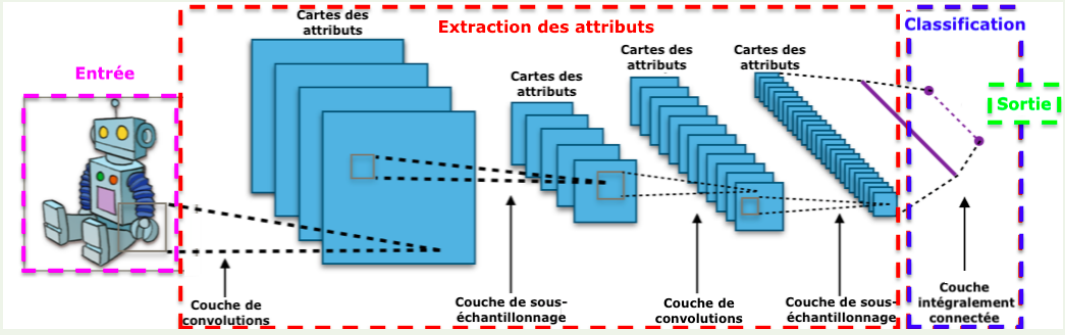


K plus proche voisins



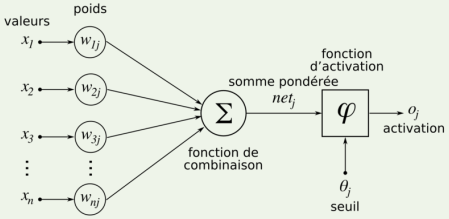
Rappel IA : Exemple : le réseau de neurones convolutifs

Architecture standard d'un réseau à convolutions

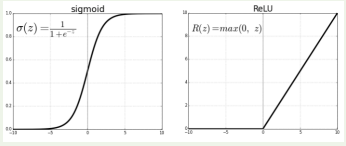


Rappel IA : Les réseaux de neurones, comment ça marche ?

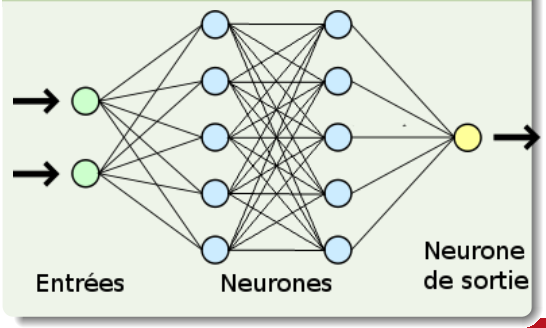
Fonctionnement d'un neurone



Example de fonctions d'activations



Architecture d'un réseau de neurones



Rappel IA : Utilisation d'un réseau de neurones

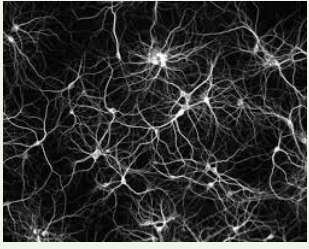
Construction d'un réseau de neurones -> Learning

- Confection d'un dataset de données préalablement classé
- Création de l'architecture du réseau
- Apprentissage du modèle
  - Exécution du modèle sur les données d'apprentissages
  - Ajustement des poids du modèle en fonction du résultat (Rétropropagation)
- Validation du modèle

Déploiement d'un réseau de neurones -> Inférence

- Exécution de l'inférence du réseau
- Prédiction sur des données inconnues

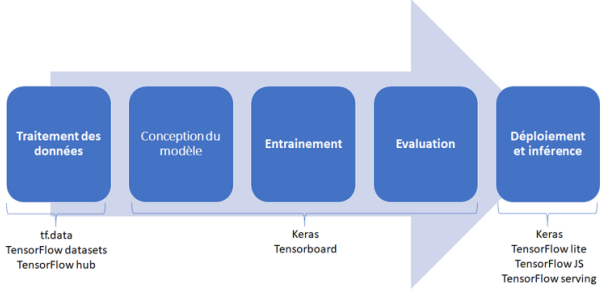
Réseau de neurones



Tensorflow : Qu'est-ce que c'est ?

Caractéristiques de Tensorflow

- Interface de développement haut niveau
- Plusieurs langages possibles
- Compatibilité multiplateforme
- Grande documentation
- Beaucoup d'outils associés

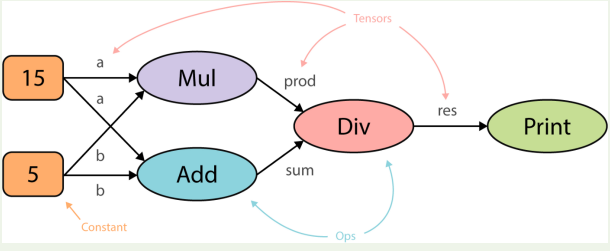


Tensorflow : Création d'un réseau de neurones

Implémentation d'un réseau de neurones sous Tensorflow

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

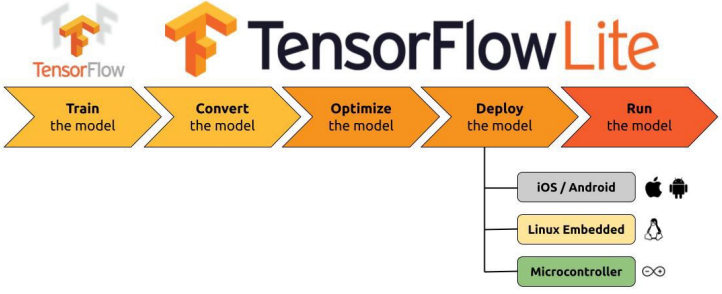
Représentation graphique d'un réseau de neurones



Tensorflow lite : Pourquoi ce choix ?

Les intérêts de TensorFlow lite pour microcontrôleurs

- Exécution uniquement de l'inférence
- Léger
- Codé en C++
- Possibilité d'implémenter des opérations optimisées
- Possibilité de compiler pour une cible spécifique



Tensorflow lite : Organisation de la bibliothèque C++

Le noyau generique de Tensorflow Lite

- Contient le code générique des opérations présentes dans le framework
- Les fonctions sont appelé lors de l'exécution de l'inférence

Le noyau pour microcontrôleur

- Appel la fonction du noyau générique en fonction des paramètres (ex : types de données)
- Contient des fonctions génériques et des fonctions optimisées pour des architectures spécifiques
- Possibilité de rajouter des opérations optimisées et adapté pour la C-SRAM

Résultat du profiling de la détection de personnes

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 81.84      0.54     0.54      112     4.82     4.82  tflite::reference_integer_ops::ConvPerChannel(tflite::ConvParams con
 18.19      0.66     0.12      112     1.07     1.07  tflite::reference_integer_ops::DepthwiseConvPerChannel(tflite::Depth
  0.00      0.66     0.00    21905     0.00     0.00  flatbuffers::Table::GetOptionalFieldOffset(unsigned short) const
```

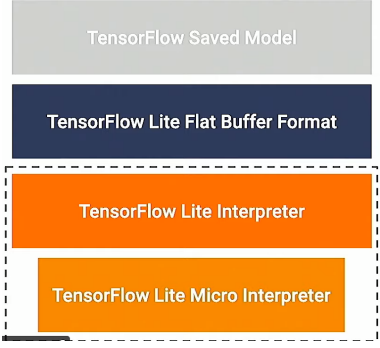
Tensorflow lite : Exemple d'un opérateur générique : La convolution

```
for (int batch = 0; batch < batches; ++batch) {
  for (int out_y = 0; out_y < output_height; ++out_y) {
    const int in_y_origin = (out_y * stride_height) - pad_height;
    for (int out_x = 0; out_x < output_width; ++out_x) {
      const int in_x_origin = (out_x * stride_width) - pad_width;
      for (int out_channel = 0; out_channel < output_depth; ++out_channel) {
        auto group = out_channel / filters_per_group;
        float total = 0.f;
        for (int filter_y = 0; filter_y < filter_height; ++filter_y) {
          const int in_y = in_y_origin + dilation_height_factor * filter_y;
          for (int filter_x = 0; filter_x < filter_width; ++filter_x) {
            const int in_x = in_x_origin + dilation_width_factor * filter_x;

            // Zero padding by omitting the areas outside the image.
            const bool is_point_inside_image =
                (in_x >= 0) && (in_x < input_width) && (in_y >= 0) &&
                (in_y < input_height);

            if (!is_point_inside_image) {
              continue;
            }
            for (int in_channel = 0; in_channel < filter_input_depth;
                 ++in_channel) {
              float input_value =
                  input_data[Offset(input_shape, batch, in_y, in_x,
                                    in_channel + group * filter_input_depth)];
              float filter_value = filter_data[Offset(
                  filter_shape, out_channel, filter_y, filter_x, in_channel)];
              total += (input_value * filter_value);
            }
          }
        }
        float bias_value = 0.0f;
        if (bias_data) {
          bias_value = bias_data[out_channel];
        }
        output_data[Offset(output_shape, batch, out_y, out_x, out_channel)] =
            ActivationFunctionWithMinMax(total + bias_value,
                                         output_activation_min,
                                         output_activation_max);
      }
    }
  }
```

Tensorflow lite : Fonctionnement de l'Inférence

Instance de la Classe MicroInterpreter

- Initialisation
  - Ajout du modèle entrainer préalablement avec Tensorflow
  - Chargement des opérations nécessaires à l'inférence
  - Création du graphe à partir du modèle
- Ajout du tenseur d'entrée
- Exécution de l'inférence avec la fonction Invoke
- Récupération des sorties



Conclusion : Les étapes pour l'implémentation des opérateurs optimisés

1. Comprendre les spécificités de la cible (ici : la C-SRAM)
   - Vectorisation
   - Transferts de données
2. Comprendre le fonctionnement de Tensorflow Lite
3. Identifier l'emplacement pour les modifications
4. Choisir les opérations les plus pertinentes à optimiser en fonction de l'application
   - Connaissance de l'application
   - Profiling d'exemples
   - Analyses des modèles
5. Implémenter les optimisations
6. Évaluer les modifications

Avez-vous des questions?