1 Préliminaires

Finissez le TP précédent : implémentez le traitement temps réel sur les images provenant de la caméra du Jetson, avec votre CNN entraîné, en vérifiant que la reconnaissance des chiffres manuscrits donne des résultats corrects.

2 TensorBoard

TensorBoard is a tool for providing the measurements and visualizations needed during the machine learning workflow.
En instrumentant un minimum votre code TensorFlow, vous pouvez fournir des données à TensorBoard qui va s'occuper de les mettre en forme et les présenter de façon conviviale via un serveur web.

2.1 Exemple 1

Dans le sous-répertoire `mnist`, étudiez le fichier `mnist_tensorboard1.py`. Vous remarquez, entre autres, que le flux de données sera mis à disposition de TensorBoard dans le répertoire `/tmp/tensorflow/mnist`. Lancez ensuite le script :

```
$ export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/usr/local/cuda/extras/CUPTI/lib64"
$ sudo rm -rf /tmp/tensorflow/mnist
$ sudo python3 mnist_tensorboard1.py
```

Et après quelques dizaines de secondes, lancez, dans un deuxième terminal :

```
$ tensorboard --logdir=/tmp/tensorflow/mnist
```

Enfin, lancez votre navigateur web et connectez-vous sur `http://localhost:6006` . Naviguez dans les éléments que TensorBoard vous propose.

2.2 Exemple 2

Il serait intéressant, dans l'exemple du CNN entraîné sur MNIST, de pouvoir visualiser le contenu des noyaux de convolution afin de voir si sémantiquement des éléments peuvent apparaître. Téléchargez, dans le sous-répertoire `mnist`, l'exemple suivant :

```
$ wget http://bit.do/fNcgc -O CNN_TB_MNIST_Example.py
```

Étudiez le script, puis lancez-le :

```
$ sudo rm -rf ./mnist_TB_logs
$ export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/usr/local/cuda/extras/CUPTI/lib64"
$ sudo python3 CNN_TB_MNIST_Example.py
```

Une fois le modèle entraîné, lancez TensorBoard :

```
$ tensorboard --logdir=./mnist_TB_logs
```

Et avec votre navigateur, connectez-vous sur `http://localhost:6006` . Visualisez les résultats d'entraînement des noyaux de convolution.

3 Cortex Visuel

3.1 Contexte

Nous allons construire un réseau de neurones convolutifs afin de reconnaître des vêtements. Premièrement, ça change des chiffres manuscrits. Deuxièmement, il existe un jeu de données *Fashion MNIST* dont le but est de remplacer celui de l'écriture manuscrite. Il est intégré, comme précédemment, dans TensorFlow.
Un réseau de neurones convolutifs, et plus spécifiquement sa couche convolutive, permet d'appliquer en parallèle plusieurs filtres de convolution sur un même support afin d'extraire des caractéristiques spécifiques, dans le but d'identifier ou retrouver ces mêmes caractéristiques dans de nouvelles images. Les paramètres de chaque noyau de convolution sont déterminés lors de la phase d'apprentissage.
Installez deux dépendances dont nous aurons besoin ultérieurement :

```
$ sudo apt install graphviz
$ sudo pip3 install pydot
```

3.2 Construction du cortex

Un réseau de neurones convolutifs est composé de deux parties principales :

- la première consiste à utiliser les différentes couches de convolution pour extraire des caractéristiques typiques des données d'entrée et ainsi obtenir une carte convolutive ;
- la seconde est composé d'un réseau de neurones classique (MLP : *Multi Layers Perceptron*) auquel sont appliquées en entrée les caractéristiques extraites par les différentes convolutions.

Nous allons construire un réseau de neurones convolutif très basique représenté sur la figure 1 :

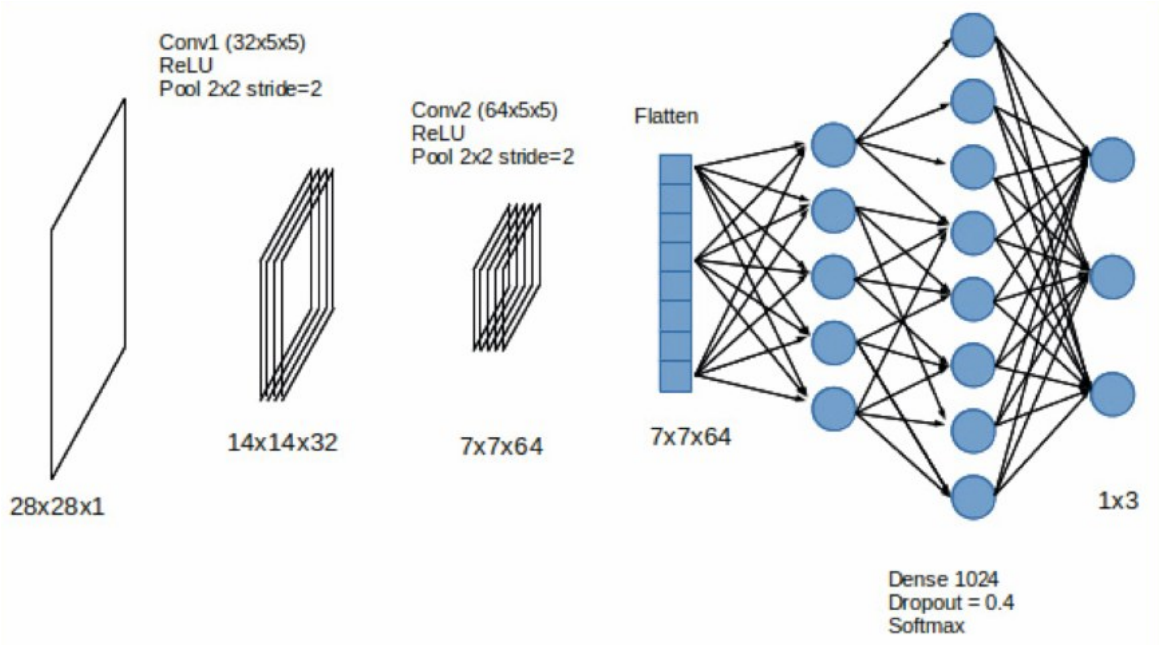

FIGURE 1 – Représentation du réseau de neurones convolutifs

Créez-vous un répertoire de travail dans lequel vous mettrez vos scripts, par exemple `cortex` .

3.2.1 Initialisation

Dans votre répertoire de travail, créez un sous-répertoire `model` dans lequel vous créez un fichier nommé `simpleCNN.py` contenant les *imports* :

```
1  from keras.models import Sequential
2  from keras.layers.normalization import BatchNormalization
3  from keras.layers.convolutional import Conv2D
4  from keras.layers.convolutional import MaxPooling2D
5  from keras.layers.core import Activation
6  from keras.layers.core import Flatten
7  from keras.layers.core import Dropout
8  from keras.layers.core import Dense
9  from keras import backend as K
```

Puis une petite fonction `simplecnn` pour créer notre modèle :

```
10 def simplecnn():
11   model = Sequential()
```

3.2.2 Couches convolutives

Concernant les couches convolutives, outre la taille du noyau de convolution, deux autres paramètres sont importants quand nous définissons une fonction de convolution : le *stride* (pas duquel est déplacé le champ récepteur à chaque itération), et le *padding* (agrandir l'image en ajoutant des 0 pour appliquer la convolution sur l'image entière).
Ajoutons une première couche convolutive (taille image entrée : 28x28, nombre de noyaux de convolution : 32, taille des noyaux : 5x5), et la fonction d'activation à la sortie de la couche convolutive (dont le but est de casser la linéarité) :

```
12     model.add(Conv2D(filters=32, kernel_size=(5,5),
13                      padding='same', input_shape=(28,28,1)))
14     model.add(Activation("relu"))
```

Et la seconde couche, conformément au schéma :

```
15     model.add(Conv2D(filters=32, kernel_size=(5,5),
16                      padding='same', activation='relu'))
17     model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
18     model.add(Dropout(0.2))
```

MaxPooling2D : couche de *pooling* utilisant l'algorithme Max, *ie* seule la valeur maximale sera retenue pour un champ récepteur (2,2) et un pas de 2 vertical et horizontal. Cela a pour effet de réduire la taille des données de sortie.

Dropout : on n'utilise que 80% des neurones de la couche suivante, afin de minimiser le surapprentissage. Ainsi, aucun neurone ne se spécialise dans une fonction.

Ajoutons les couches supplémentaires ; celles-ci ont un nombre de filtres plus important afin d'extraire des caractéristiques plus complexes :

```
19      model.add(Conv2D(filters=64, kernel_size=(5,5),
20                       padding='same', activation='relu'))
21      model.add(Conv2D(filters=64, kernel_size=(5,5),
22                       padding='same', activation='relu'))
23      model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
24      model.add(Dropout(0.2))
```

3.2.3 Réseau de neurones classique

En sortie de notre couche convolutive, nous utilisons un réseau de neurones traditionnel, dont le rôle sera d'effectuer la classification grâce aux différentes caractéristiques extraites par les couches convolutives.
Il est donc nécessaire d'aplatir la sortie de la couche convolutive pour former le vecteur d'entrée du réseau classique. Ensuite, nous pouvons ajouter les deux couches cachées, de 120 et 84 neurones, avec fonction d'activation de type ReLU :

```
25      model.add(Flatten())
26      model.add(Dense(120, activation='relu'))
27      model.add(Dense(84, activation='relu'))
28      model.add(Dense(10, activation='softmax'))
29      return model
```

L'objectif de notre réseau est la reconnaissance d'image parmi 10 classes, c'est pourquoi nous ajoutons ci-dessus une dernière couche, de sortie, de 10 neurones, avec activation *softmax* (probabilité d'appartenance pour chaque sortie, somme à 1).

3.2.4 Visualisation de l'architecture du réseau

La méthode `summary()` du modèle permet d'obtenir une description texte :

```
$ python3
>>> from model import simpleCNN
>>> model = simpleCNN.simplecnn()
>>> model.summary()
```

On peut aussi utiliser la fonction `plot_model` de `keras.utils` :

```
>>> from keras.utils import plot_model
>>> plot_model(model, to_file="model.png")
```

3.3 Entraînement du cortex

3.3.1 Compilation du modèle

Créez un fichier `train.py` dans votre répertoire de travail, dans lequel on définira toutes les étapes de l'apprentissage :

```
1  from keras.utils import np_utils
2  from keras import backend as K
3  from keras.optimizers import Adam
4  from model import simpleCNN
5
6  model = simpleCNN.simplecnn()
7  model.compile(optimizer="adam",
8                loss="categorical_crossentropy",
9                metrics=["accuracy"])
```

On compile le modèle, en précisant le type d'optimisation souhaité, la fonction de perte utilisée, ainsi que la méthode d'évaluation du résultat.

3.3.2 Importation du jeu de données

Le jeu de données *Fashion MNIST* est composé de plusieurs catégories de vêtements ; ajoutons donc une liste des noms et importons les données depuis Keras :

```
10 data_label = ["t-shirt", "pantalon", "pull-over", "robe",
11               "manteau", "sandale", "chemise", "basket",
12               "sac", "bottine"]
13
14 from keras.datasets import fashion_mnist
15 ((trainX,trainY), (testX,testY)) = fashion_mnist.load_data()
```

3.3.3 Préparation des données

Pour que les données soient compatibles avec notre modèle, nous devons les restructurer. Nous venons de charger des données sous la forme `nb données x largeur x hauteur` et notre modèle attend en entrée une information supplémentaire concernant la profondeur, appelée également canal. Avec la fonction `reshape`, réordonner les données :

```
16 if K.image_data_format() == "channels_first":
17   trainX = trainX.reshape((trainX.shape[0], 1, 28, 28))
18   testX = testX.reshape((testX.shape[0], 1, 28, 28))
19 else:
20   trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
21   testX = testX.reshape((testX.shape[0], 28, 28, 1))
```

Normalisons ensuite les données (même plage pour toutes les données, et stabilité lors des calculs des paramètres du réseau), et passons les données de sortie en encodage *one-hot* :

```
22  trainX = trainX.astype("float32") / 255.0
23  testX = testX.astype("float32") / 255.0
24  trainY = np_utils.to_categorical(trainY, 10)
25  testY = np_utils.to_categorical(testY, 10)
```

3.3.4 Apprentissage

Lançons l'apprentissage avec la méthode `fit`, en spécifiant le nombre d'itérations (`epochs`), la taille des lots (`batch_size`) :

```
26  hist = model.fit(trainX, trainY,
27                   validation_data=(testX, testY),
28                   batch_size=16, epochs=10)
```

Ajustez le `batch_size` de manière à ce qu'il n'y ait pas d'erreur de mémoire sur le Jetson.

Évaluons notre modèle sur un jeu de données test avec la méthode `evaluate()` de Keras :

```
29  model.evaluate(testX, testY)
```

Comparez votre résultat au modèle **VGG16**.

3.3.5 Évolution de l'apprentissage

La variable `hist`, retournée par la fonction `fit()`, contient les statistiques sur les différentes itérations de l'apprentissage. On peut tracer graphiquement l'évolution de l'apprentissage :

```
30  from matplotlib import pyplot as plt
31  plt.plot(hist.history['acc'])
32  plt.plot(hist.history['val_acc'])
33  plt.title('Precision du modele')
34  plt.ylabel('Precision')
35  plt.xlabel('Iteration')
36  plt.legend(['Apprentissage', 'Test'], loc='upper left')
37  plt.show()
```

3.3.6 Sauvegarde du modèle

Keras propose tout ce qu'il faut pour enregistrer le modèle :

```
38  from keras.models import load_model
39  model.save('model.h5')
```

Et le recharger :

```
1  from keras.models import load_model
2  model = load_model('model.h5')
```

3.4 Tests et validation

3.4.1 À partir des données de test

Nous allons prendre quelques images aléatoirement parmi les données de test.
Créez un fichier `testmodel.py` :

```
1  import numpy as np
2  from keras.datasets import fashion_mnist
3  from keras import backend as K
4  from keras.utils import np_utils
5  from matplotlib import pyplot as plt
6
7  data_label = ["t-shirt", "pantalon", "pull-over", "robe",
8                "manteau", "sandale", "chemise", "basket",
9                "sac", "bottine"]
10
11 ((trainX,trainY), (testX,testY)) = fashion_mnist.load_data()
12
13 if K.image_data_format() == "channels_first":
14   testX = testX.reshape((testX.shape[0], 1, 28, 28))
15 else:
16   testX = testX.reshape((testX.shape[0], 28, 28, 1))
17
18 testX = testX.astype("float32") / 255.0
19 testY = np_utils.to_categorical(testY, 10)
```

La méthode `predict()` du modèle permet de prendre un tableau en entrée (une image), et de sortir la probabilité d'appartenance de l'image à chaque classe.

```
20  tabimages=[]
21  rand_indexes = np.random.choice(np.arange(0, len(testY)),
22                                  size=(16,))
23  for i in rand_indexes:
24    results = model.predict(testX[np.newaxis, i])
25    prediction = results.argmax(axis=1)
26    label = data_label[prediction[0]]
27    if K.image_data_format() == "channels_first":
28      image = (testX[i][0]*255).as_type('uint8')
29    else:
30      image = (testX[i]*255).as_type('uint8')
31
32    couleurTXT=(0,255,0)
33    if prediction[0] != np.argmax(testY[i]):
34      couleurTXT=(255,0,0) # mauvaise prediction
35
36    image = cv2.merge([image] * 3)
37    cv2.putText(image, label, (5,20), cv2.FONT_HERSHEY_SIMPLEX,
38                0.75, couleurTXT, 2)
39    tabimages.append(image)
40
41  plt.figure(figsize=(7,7))
42  for i in range(0,len(tabimages)):
43    plt.subplot(4,4,i+1)
44    plt.imshow(tabimages[i])
45  plt.show()
```

Quels sont les résultats ?

3.4.2 Avec d'autres sources d'images

— Naviguez sur des sites comme `https://www.sarenza.com/`, récupérez des images de vêtements et écrivez un code python permettant d'interroger votre modèle sur l'image. Quels sont les problèmes que vous pouvez déjà anticiper ?
— Utilisez la caméra du Jetson également.