Utilisation de l'apprentissage profond dans la reconnaissance d'espèces sous-marines

Code source en Bash et Python, appuyé sur TensorFlow

Lucas Tabary

1 Traitement des données

Les données récupérées de l'ensemble *Fish4Science* ne sont pas sous une présentation convenable. Il convient de les traiter, en les déplacement et le renommant convenablement. Les images sont soit de vraies photos, soit des masques binaires (qui permettent de de dévoiler la forme du poisson sur l'image). Il convient de déplacer les images dans un nouveau dossier, en créant un fichier CSV regroupant les couples (ID de la photo, numéro de l'espèce). Le nouveau dossier formera alors la séparation images de test *(test)* et d'entraînement *(train)*.

```
1  #!/bin/bash
2  # Structure originale : <fish/mask>_image/<fish/mask>_<no_espece>/
       <fish/mask>_<no_inutile>_<id_image>.png
3  # Structure souhaitée : <test/train>/<id_image><m/i>.png + fichier CSV
       indiquant les correspondances ID <-> espèce
4
5  mkdir -p images images/test images/train
6
7  for folder in fish_image/*; do
8    if [[ -d $folder ]]; then
9      species=$(echo $folder | sed -r 's/.*_0*([0-9]*)$/\1/g') # Numéro de
           l'espèce
10     echo "Traitement␣de␣l'espèce␣$species"
11     for image in ${folder}/*; do
12       maskfile=$(echo $image | sed -r 's/fish/mask/g') # masque associé à
             l'image
13       fish_id=$(echo $image | sed -r 's/.*_0*([0-9]*)\.png/\1/g') # ID de
             l'image
14       destination="train"
15       if grep -Fxq "$fish_id" test_index; then # Si l'ID de l'image fait parti
             des images de test
16         destination="test"
17       fi
18       mv $image images/${destination}/${fish_id}i.png
19       mv $maskfile images/${destination}/${fish_id}m.png
20       echo "${fish_id},␣${species}" >> images/${destination}_labels.csv
21     done
22   fi
23 done
```

Figure 1 – Déplacement et renommage des fichiers – `name_formatting.sh`

On cherche ensuite à rogner toutes les images pour obtenir un format unique de 100 par 100 pixels, afin de pouvoir fournir les données au réseau (qui sera construit pour recevoir des images de telle dimension).

```python
from PIL import Image
import glob

working_directories = ['images/train', 'images/test']

for directory in working_directories:
    for image in glob.glob(directory + "/*.png"):
        im = Image.open(image)
        width, height = im.size
        if width == 100 and height == 100: # L'image est déjà dimensionnée
            continue
        new = min(width, height)
        left = (width - new) / 2
        top = (height - new) / 2
        right = (width + new) / 2
        bottom = (height + new) / 2

        im_new = im.crop((left, top, right, bottom)).resize((100, 100))
        im_new.save(image, 'PNG')
```

FIGURE 2 – Rognage des différentes images – `image_formatting.py`

2 Création du module d'acquisition des données

On crée une interface (un module PYTHON) permettant l'acquisition des données sous une forme utilisable par le réseau (en l'occurence, des `numpy.array`s).

```python
from PIL import Image
import numpy as np
import random
import pickle

PART_OF_TEST = 0.1
NUM_IMAGES = 27370
NUM_TRAIN_IMAGES = int(PART_OF_TEST * NUM_IMAGES)
NUM_TEST_IMAGES = NUM_IMAGES - NUM_TRAIN_IMAGES

def generate_test_index(path='fishdataset/test_index'):
    """Génère une liste aléatoire d'IDs de photos pour les séparer en deux groupes test/train.
       Son utilisation doit être couplée à la génération des fichiers via 'name_formatting.sh'."""
    INDEX = random.sample(range(1, NUM_IMAGES + 1), int(PART_OF_TEST * NUM_IMAGES))
    with open(path, 'w') as file:
        for elt in INDEX:
            file.write(str(elt) + '\n')

def acquire_dataset(split='train'):
    """Renvoie deux `numpy.array`s contenant les ensembles des images et des étiquettes."""
    try: # Récupère s'ils existent les arrays déjà construits
        with open('fishdataset/images/' + split + '_images.pickle', 'rb') as f:
            images = pickle.load(f)
        with open('fishdataset/images/' + split + '_labels.pickle', 'rb') as f:
            labels = pickle.load(f)
        return images, labels
    except FileNotFoundError:
        pass
```

```
29
30      ids, labels = [], []
31      with open('fishdataset/images/' + split + '_labels.csv') as csv_file:
32          for line in csv_file.readlines():
33              id, label = line.strip().split(',', maxsplit=2)
34              ids.append(id)
35              labels.append(int(label) - 1)
36
37      # `mapping` des valeurs du dataset pour obtenir les véritables couples
            (tenseur, étiquette)
38      def associate_image(id):
39          path = 'fishdataset/images/' + split + '/' + id
40
41          image = Image.open(path + 'i.png')
42          mask = Image.open(path + 'm.png')
43
44          image_array = np.array(image)
45          mask_array = np.array(mask)
46
47          image = np.dstack((image_array, mask_array)) # Concatène les array sur
                la 4e dimension (canaux RGB-Masque)
48          return image
49
50      images = np.array([associate_image(id) for id in ids], dtype=np.float16)
51      labels = np.array(labels)
52
53      # Sauvegarde les arrays pour réutilisation
54      with open('fishdataset/images/' + split + '_images.pickle', 'wb') as f:
55          pickle.dump(images, f)
56      with open('fishdataset/images/' + split + '_labels.pickle', 'wb') as f:
57          pickle.dump(labels, f)
58
59      return images, labels
```

FIGURE 3 – Module d'acquisition des données – `fishdataset.py`

3 Construction et entraînement du réseau

La construction du réseau se fait de manière opaque à l'aide de TENSORFLOW. On décrit l'architecture du réseau ainsi que la méthode d'optimisation (diminution de la fonction de coût). On a aussi ajouté des fonctions permettant de rapidement sauvegarder ou charger un modèle.

On a construit le graphe d'évolution de la précision et de la perte à partir du dictionnaire généré automatiquement `model.history.history`.

```
1  import fishdataset
2  from tensorflow.keras import layers, models
3
4  train_images, train_labels = fishdataset.acquire_dataset(split='train')
5  test_images, test_labels = fishdataset.acquire_dataset(split='test')
6
7  # Passage au bon format des images
8  train_images, test_images = train_images / 255.0, test_images / 255.0
9
10 model = models.Sequential()
11
12 def build_cnn():
```

```
13      model.add(layers.Conv2D(32, (5, 5), activation='relu', input_shape=(100,
            100, 4)))
14      model.add(layers.MaxPooling2D((3, 3)))
15      model.add(layers.Conv2D(64, (3, 3), activation='relu'))
16      model.add(layers.MaxPooling2D((2, 2)))
17      model.add(layers.Conv2D(64, (3, 3), activation='relu'))
18      model.add(layers.Flatten())
19      model.add(layers.Dense(100, activation='relu'))
20      model.add(layers.Dropout(0.8))
21      model.add(layers.Dense(64, activation='relu'))
22      model.add(layers.Dense(23, activation='softmax'))
23
24      model.compile(optimizer='adam',
25                    loss='sparse_categorical_crossentropy',
26                    metrics=['accuracy'])
27
28  def build_rathi():
29      model.add(layers.Conv2D(32, (5, 5), activation='relu', input_shape=(100,
            100, 4)))
30      model.add(layers.MaxPooling2D((5, 5)))
31      model.add(layers.Conv2D(64, (5, 5), activation='relu', padding='same'))
32      model.add(layers.MaxPooling2D((5, 5)))
33      model.add(layers.Conv2D(64, (5, 5), activation='relu', padding='same'))
34      model.add(layers.Flatten())
35      model.add(layers.Dense(100, activation='relu'))
36      model.add(layers.Dropout(0.8))
37      model.add(layers.Dense(23, activation='softmax'))
38
39      model.compile(optimizer='adam',
40                    loss='sparse_categorical_crossentropy',
41                    metrics=['accuracy'])
42
43  def build_multilayer_perceptron():
44      model.add(layers.Flatten())
45      model.add(layers.Dense(1000, activation='relu'))
46      model.add(layers.Dense(500, activation='relu'))
47      model.add(layers.Dense(100, activation='relu'))
48      model.add(layers.Dense(23, activation='softmax'))
49
50      model.compile(optimizer='adam',
51                    loss='sparse_categorical_crossentropy',
52                    metrics=['accuracy'])
53
54  def train_model(epochs=10):
55      model.fit(train_images, train_labels, epochs=epochs)
56
57  def test_model():
58      test_loss, test_acc = model.evaluate(test_images, test_labels)
59      print(test_acc)
60
61  def save_model(path='state.h5'):
62      model.save(path)
63
64  def retrieve_model(path='state.h5'):
65      return models.load_model(path)
```

FIGURE 4 – Fichier principal – `main_cnn.py`