ENSG Géomatique

ÉCOLE NATIONALE
DES SCIENCES
GÉOGRAPHIQUES

Ecole Nationale des
Sciences Géographiques

Cycle des ingénieurs diplômés de l'ENSG 3ème année

Appariement de points d'intérêt par Deep Learning
Guide utilisateur

Guillemette Fonteix

Commanditaires : Ewelina Rupnik & Marc Pierrot-Deseilligny

Février 2019

☒ Non confidentiel ☐ Confidentiel IGN ☐ Confidentiel Industrie ☐ Jusqu'au ...

Ecole Nationale des Sciences Géographiques
6-8 Avenue Blaise Pascal - Cité Descartes - 77420 Champs-sur-Marne
Téléphone 01 64 15 31 00 Télécopie 01 64 15 31 07

Table des matières

Chapitre 1
Introduction

Dans le cadre des projets informatiques des étudiants de PPMD de l'ENSG, Ewelina Rupnik et Marc Pierrot-Deseilligny (LaSTIG, IGN) ont proposé un sujet se rattachant à l'appariement de points d'intérêt dans les images. Le but de ce projet est de trouver une méthode alternative à l'algorithme SIFT afin de retrouver les points homologues par Deep Learning. Ce rapport utilisateur a pour objectif de comprendre comment lancer le code pour la découpe d'images, comment lancer l'apprentissage du réseau neuronal sur nos données ou sur de nouvelles données et enfin comprendre comment réutiliser ce modèle. Il est complété par le rapport sur la phase de programmation et le guide du développeur. Il est conseillé d'avoir lu le rapport sur la phase de développement avant celui-ci.

Notons que ce guide n'est pas conséquent puisque l'utilisateur est aussi le développeur, le but étant de fournir un code prototype de Deep Learning pour l'appariement de points d'intérêt à mes commanditaires qui le réutiliseront par la suite (ils seront alors utilisateurs et développeurs).

Chapitre 2
Les données

2.1 Comment les données ont été découpées ?

Il m'a été fourni pour chaque point caractéristique (GrayMax, GrayMin, LaplMin, LaplMax, BifurqMin, BifurqMax) et pour les trois descripteurs calculés pour chaque point caractéristique (ACR0, ACGT, ACGR) des images au format .tif associant des descripteurs de points homologues pour chaque paire d'images.

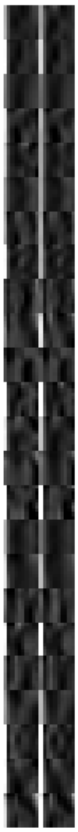

Figure 2.1 – Ensemble de descripteurs de points homologues pour un couple d'images

Pour chaque .tif, j'ai découpé les descripteurs de points homologues de taille 10x20 pixels pour les descripteurs ACR0 et ACGT et 10x18 pixels pour le descripteur ACGR. J'ai créé deux dossiers, l'un avec des couples de descripteurs homologues :

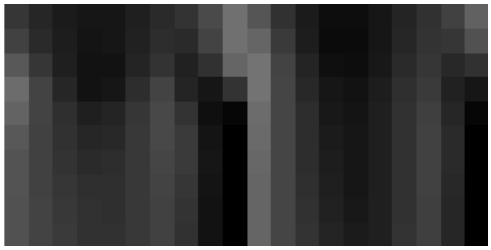

FIGURE 2.2 – Exemple couple de descripteurs de points homologues

L'autre avec des descripteurs de points non-homologues :

FIGURE 2.3 – Exemple couple de descripteurs points non-homologues

Les images sont rangées selon l'arborescence suivante :

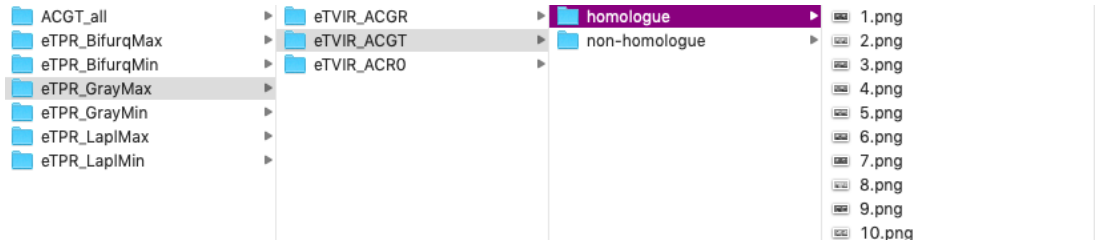

FIGURE 2.4 – Arborescence des dossiers contenant les images

2.2 Comment découper de nouvelles images ?

Si vous souhaitez découper de nouvelles images, il suffit d'utiliser les scripts **decoupe_homologue** pour découper les descripteurs de points homologues et **decoupe_non_homologue** pour découper les descripteurs de points non-homologues.

Comment faire ?

1. Placer ce script dans un dossier vide.
2. Changer le chemin vers vos images .tif (path).
3. Vous pouvez aussi changer le nom donné à vos images lorsque vous enregistrez les images (imsave). J'ai choisi de les nommer par des nombres (on incrémente à chaque itération pour ne pas écraser les données existantes).
4. Si vos données doivent être découpées en 10x20 : lancer le programme.

Sinon il va falloir modifier la taille des images pour s'adapter à vos données en changeant les arguments des fonctions.

Chapitre 3 — Comment faire fonctionner le programme de Deep Learning ?

3.1 Comment entraîner les données ?

Dans le script **CNN_points_homologues**, il suffit de changer le chemin vers l'endroit où se trouvent vos données (images découpées et triées dans deux dossiers *homologue* et *non-homologue* : voir figure 2.4). Vous pouvez alors lancer le programme.

Vous pourrez aussi, si vous le souhaitez, rentrer manuellement la moyenne et l'écart type pour la normalisation en fonction du type de données afin de gagner du temps (voir guide du développeur).

3.2 Qu'est-ce que j'obtiens en sortie ?

• En sortie de l'algorithme, nous obtiendrons tout d'abord le nombre d'images contenu dans les données d'entraînement, de validation et de test.

```
There are 48000 images in the training set
There are 16000 images in the test set
There are 16000 images in the validation set
Loading completed successfully!
```

FIGURE 3.1 – Chargement des données d'entraînement, de validation et de test

• Avant l'entraînement, on va tester le modèle avec les données de test. On obtient une valeur proche de 50 % puisque notre problème est binaire et que le modèle n'est pas encore entraîné.

```
Test Accuracy of the model on the  test images: 49.88125 %
```

FIGURE 3.2 – Test avant l'entraînement sur les données de test

• A chaque époque, on regarde la perte et on affiche la précision sur les données de validation.

```
Epoch [1/15], Step [100/1200], Loss: 0.3723
Epoch [1/15], Step [200/1200], Loss: 0.5774
Epoch [1/15], Step [300/1200], Loss: 0.2509
Epoch [1/15], Step [400/1200], Loss: 0.2674
Epoch [1/15], Step [500/1200], Loss: 0.1807
Epoch [1/15], Step [600/1200], Loss: 0.3102
Epoch [1/15], Step [700/1200], Loss: 0.2204
Epoch [1/15], Step [800/1200], Loss: 0.1411
Epoch [1/15], Step [900/1200], Loss: 0.2716
Epoch [1/15], Step [1000/1200], Loss: 0.1240
Epoch [1/15], Step [1100/1200], Loss: 0.1988
Epoch [1/15], Step [1200/1200], Loss: 0.1270
Valid Accuracy of the model on the validation set: 93.58125 %
```

FIGURE 3.3 – Exemple valeur de la fonction de perte sur une époque et précision grâce aux données de validation

• Nous affichons ensuite la fonction de perte. Cette fonction doit être minimisée et doit donc être décroissante au fil des époques.

• Lorsque l'apprentissage est terminé, nous évaluons notre modèle avec les données test (qui n'ont pas été utilisé pendant l'apprentissage). On obtient alors une précision de notre algorithme.

```
Test Accuracy of the model on the  test images: 94.675 %
```

FIGURE 3.4 – Test après l'entraînement sur les données de test

• On affiche ensuite la matrice de confusion, le but étant d'avoir un score proche de 1 sur la diagonale.

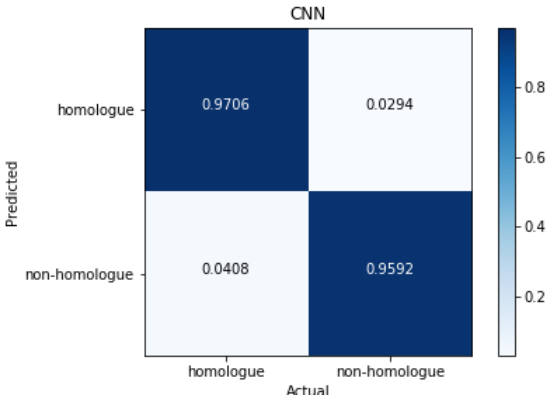

FIGURE 3.5 – Exemple de matrice de confusion

• On affiche enfin les courbes ROC. Pour plus d'explications, se référer au rapport sur la phase de développement.

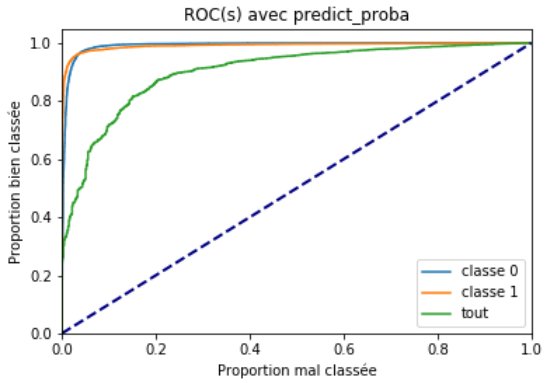

FIGURE 3.6 – Exemple de courbes ROC

3.3 Comment réutiliser le modèle entraîné sur de nouvelles données ?

Suite à l'apprentissage, un modèle pourra être enregistré et permettra de réutiliser le CNN qui a précédémment appris sur les données. On pourra alors donner de nouvelles images sans label (on se demande si celles-ci sont homologues ou non-homologues sans en avoir la réponse).

Vous pourrez choisir l'endroit où le modèle sera enregistré dans la partie 'parameters and paths' (path_model). Un modèle a précédemment été enregistré pour vous sur les points caractéristiques GrayMax pour le descripteur ACGT. Vous pourrez donc réutiliser directement ce modèle (*model.ckpt*) sans réaliser l'apprentissage si vous le souhaitez. Vous pourrez alors commenter le code de l'apprentissage pour ne pas le lancer à nouveau.

Il suffira alors de lancer la fonction *reutilisation(path)* avec comme argument le chemin vers l'image à classifier.

Cette fonction indiquera simplement si votre image correspond à un couple de descripteurs de points homologues ou non-homologues (avec une certaine précision qui correspond à la précision de l'apprentissage).

Table des figures