ENSG Géomatique

ÉCOLE NATIONALE
DES SCIENCES
GÉOGRAPHIQUES

Ecole Nationale des
Sciences Géographiques

Projet informatique

Cycle des ingénieurs diplômés de l'ENSG 3ème année

Appariement de points d'intérêt par Deep Learning
Rapport sur la phase de développement

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
Quelques rappels

1.1 Contexte

L'appariement de points d'intérêt dans les images est la première étape dans la chaîne de traitement photogrammétrique. En photogrammétrie, SIFT (Scale Invariant Feature Transform) est l'algorithme typiquement utilisé. Il est invariant aux rotations, à l'échelle et partiellement aux transformations affines. Cependant, son temps de calcul est long et sa licence d'utilisation est contraignante. C'est pourquoi, pour parer à ces désavantages et dans le cadre du développement d'un nouvel algorithme d'extraction (détection et appariement) de points homologues (AIME) au sein de l'IGN, il m'a été demandé d'utiliser des méthodes de Deep Learning afin d'apparier des points d'intérêt dans de grands jeux de données.
Les algorithmes de détection des points d'intérêt (détails remarquables dans les images, correspondant à des doubles discontinuités de la fonction d'intensité) ont déjà été implémentés. De plus, des descripteurs caractérisant ces points ont été calculés.

1.2 Problématique

Etant donné un point d'intérêt (i_1, j_1) de l'image 1 (image de référence), on peut retrouver son point homologue (i_2, j_2) (point image correspondant au même détail terrain) dans l'image 2 (image secondaire). En effet en intersectant (i_1, j_1) avec la géométrie de la scène, on obtient un point 3D correspondant au point d'intérêt. Puis, en reprojetant le point 3D dans l'image secondaire on aura une position (x, y) d'un point homologue.

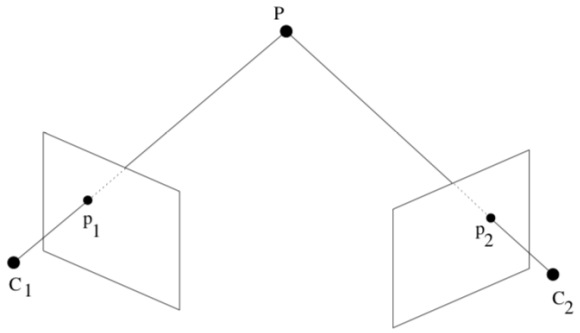

Figure 1.1 – Appariement de points d'intérêt

Cependant reste à savoir si l'implémentation d'un algorithme de Deep Learning peut être une alternative efficace pour l'appariement de ces points d'intérêt dans de grands jeux de données.

1.3 Objectif

L'objectif de ce projet est de mettre en correspondance des points d'intérêt entre différentes images par l'intermédiaire d'algorithmes de Deep Learning.

Il m'a fallu construire une architecture CNN (convolutional neural network) permettant d'apprendre et de prédire si deux points sont homologues par l'intermédiaire de leurs descripteurs.

Le but étant de répondre à la question suivante : **quel score de performance peut-on obtenir pour l'appariement de points d'intérêt dans les images grâce au Deep Learning ?**

Chapitre 2 Modification du développement par rapport à l'analyse

2.1 Les données

Au départ, je n'étais censée travailler que sur un seul type de descripteur de points caractéristiques "eTPR_GrayMax - eTVIR_ACR0" correspondant à des points caractéristiques de maximum de niveau de gris et des descripteurs calculés par corrélation de l'image avec elle-même.

Cependant, après avoir créé une première version de l'architecture CNN, il m'a été demandé d'utiliser plusieurs types de points caractéristiques et différents descripteurs. Le but est de savoir quel descripteur nous donne les meilleurs résultats pour comprendre quelles seront les meilleures données à utiliser pour l'apprentissage.

Ces données viennent de six acquisitions différentes, pour chaque acquisition il y avait une dizaine d'images.

2.1.1 Les points caractéristiques

Les différents types de points caracteristiques étudiés :
• eTPR_GrayMax => maximum du niveau de gris
• eTPR_GrayMin => minimum du niveau de gris
• eTPR_LaplMin => minimum du laplacien
• eTPR_LaplMax => maximum du laplacien
• eTPR_BifurqMin => minimum de bifurcation
• eTPR_BifurqMax => maximum de bifurcation

Pour chaque type de point, trois types de descripteurs ont été calculés. Il s'agit de déterminer quel type de descripteur obtient les meilleurs résultats avec notre réseau de neurones.

2.1.2 Les descripteurs

Les différents types de descripteurs :
• eTVIR_ACR0 => auto corrélation : on corrèle l'image avec elle même. Descripteur de taille 10x10 pixels.
• eTVIR_ACGT => corrélation de l'image avec le gradient tangentiel. Descripteur de taille 20x10 pixels.
• eTVIR_ACGR. Descripteur de taille 10x9 pixels.

Le dossier prioritaire à traiter était "eTPR_GrayMax/eTVIR_ACR0/". Il contient environ 450 000 couples (inféré du fait que la taille du dossier est de 91 Mo et que chaque couple fait 200 o). Seulement une partie des données a été utilisée pour les tests (40 000 paires de descripteurs de points homologues et 40 000 de points non-homologues).

2.2 Des questions subsidiaires

Une fois l'architecture mise en place, il m'a été demandé de répondre aux questions suivantes :

1/ Parmi les différents types de descripteurs, quel est le meilleur ?

2/ Parmi les différents types de points caractéristiques, quel est le meilleur ?

3/ Est-ce que si l'on mélange les types de points, on gagne en score de précision ?

2.3 Programme prévisionnel vs programme réel

2.3.1 Programme prévisionnel

Etape 1. Préparation des données.
Etape 2.
a. Création de notre CNN (Convolutional Neural Network).
b. Entraînement des données et test du modèle.
Etape 3.
a. Modification d'un réseau CNN pré-entraîné s'adaptant à notre problème grâce au Transfer Learning.
b. Entraînement des données et test du modèle.
Etape 4. Evaluation des algorithmes et choix du meilleur modèle.

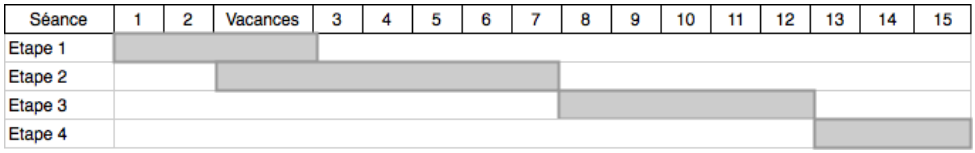

FIGURE 2.1 – Planning prévisionnel

2.3.2 Programme réel

Etape 1. Préparation des données "eTPR_GrayMax/eTVIR_ACR0/".
Etape 2.
a. Création de notre CNN.
b. Entraînement des données et test du modèle.
Etape 3. Préparation des données pour chaque point caractéristique et chaque descripteur.
Etape 4. Entraînement des données et test du modèle sur les différents descripteurs des différents points caractéristiques. Choix du meilleur descripteur à utiliser pour réaliser l'appariement.
Etape 5. Ecriture des rapports.

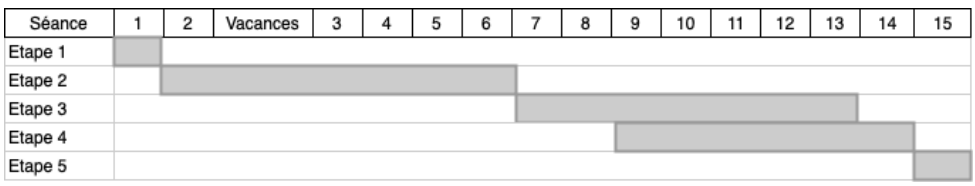

FIGURE 2.2 – Planning réel

6 Modification du développement par rapport à l'analyse

On peut remarquer que la partie sur le Transfer Learning n'a pas été réalisée. En effet, suite aux résultats encourageants de mon architecture CNN j'ai préféré me concentrer sur cette partie. De plus, après des recherches internet et des conversations avec les personnes nous aidant pour le projet, je me suis rendue compte qu'il n'y avait pas beaucoup de chance pour que le Transfer Learning nous donne des résultats satisfaisants. Je ne voulais pas perdre de temps sur cette étape, j'ai préféré améliorer les performances de mon réseau et répondre au mieux aux questions subsidiaires arrivées en milieu de projet.

Chapitre 3
Résultats

3.1 Le problème de départ

L'algorithme d'apprentissage peut se décomposer en quatre grandes parties comme indiqué sur le schéma. Premièrement, il a fallu préparer les données en créant des classes pour les points homologues et non-homologues. Puis, j'ai subdivisé les données en échantillons d'apprentissage, de validation et de test.
Ensuite, il s'agissait de créer l'architecture du réseau neuronal (choix du nombre des couches de convolutions, pooling, etc) et de choisir les différentes caractéristiques de l'algorithme d'apprentissage (fonction d'activation, fonction de perte et algorithme d'optimisation). L'étape suivante est l'entraînement des données et la validation du modèle (permettant à chaque époque d'ajuster les poids).
La dernière étape consistait à utiliser des données de test pour évaluer la performance du réseau (courbe ROC, matrice de confusion et score de précision) pour faire le choix du meilleur modèle.

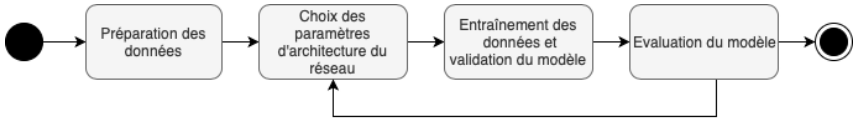

Figure 3.1 – Base de l'algorithme

3.2 Résultats

3.2.1 Etape 1. Préparation des données

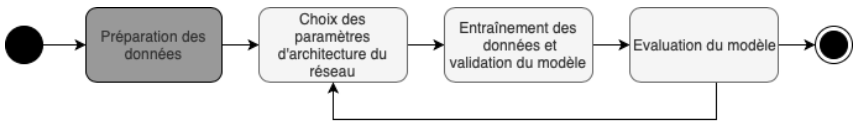

Figure 3.2 – Etape 1. Préparation des données

Découpage des images

Pour chaque type de point caractéristique et pour chaque type de descripteur, il a fallu découper les images .tif en image 10x20 pixels (pour les descripteurs ACGT et ACR0) et en image 10x18 pixels (pour les descripteurs ACGR). On a créé deux classes : homologue et non-homologue.

Transformations appliquées aux images

Avant de réaliser l'apprentissage sur le réseau de neurones, il faut appliquer quelques transformations sur les images : transformation en tenseur et normalisation (et donc calcul de la moyenne et de l'écart type pour tous les jeux de données).

3.2.2 Etape 2. Choix des paramètres d'architecture du réseau

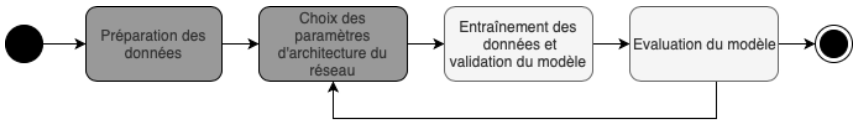

FIGURE 3.3 – Etape 2. Choix des paramètres d'architecture du réseau

J'ai donc créé une architecture CNN composée de :
• **Deux couches de convolution** avec fonction d'activation ReLU : elles vont mettre en valeur quelques caractéristiques dans les images.
• **Une couche de Pooling** (sous échantillonage) : c'est une méthode permettant de prendre une large image et d'en réduire la taille tout en préservant les informations les plus importantes qu'elle contient. L'objectif est de sous-échantillonner une image tout en conservant la valeur maximale dans les sous-régions regroupées et ne pas perdre d'information. Une seule est suffisante dans notre cas puisque nos images sont petites, il ne faut donc pas trop sous échantillonner car on perdrait l'intégralité de l'information.
• **Quatre fully-connected** : couches entièrement connectées. Elles sont les principaux blocs de construction des réseaux de neurones traditionnels. Au lieu de traiter les entrées comme des tableaux de 2 dimensions, ils sont traités en tant que liste unique et tous traités de manière identique. Chaque valeur a son propre vote pour déterminer si l'image est un homologue ou un non-homologue. Certaines valeurs sont bien meilleures à détecter lorsqu'une image est un homologue que d'autres, et d'autres sont bien meilleures à détecter un non-homologue. Celles-ci ont donc davantage de pouvoir de vote que les autres. Ce vote est appelé le poids entre chaque valeur et chaque catégorie.

J'ai donc choisi de créer un réseau qui n'est pas très profond. En effet, je n'utilise que deux couches de convolution et une couche Pooling compte tenu de la taille réduite des images en entrée (10x20 pixels).

L'un des aspects embarrassants de la définition manuelle des réseaux de neurones est la nécessité de spécifier les tailles des entrées et des sorties à chaque partie du processus. Les commentaires dans le code devraient donner une idée de ce qui se passe avec les changements de taille à chaque étape. Le réseau neuronal va donc dépendre de la taille des images en entrée. Or, selon les descripteurs, la taille des images est différente. Elle est la même pour les descripteurs ACR0 et ACGT (taille 10x20 pixels). Le descripteur ACGR est de taille 10x18 pixels. Il faudra donc changer la taille des entrées et des sorties dans notre CNN en fonction du descripteur utilisé.

En général, la taille de sortie après une convolution de n'importe quelle dimension de notre jeu d'entrées peut être définie comme suit :

$$output\_size = \frac{in\_size - kernel\_size + 2 \times (padding)}{stride} + 1$$

Il a fallu faire des choix concernant ces paramètres (taille du kernel, de la stride et du padding) pour les couches de convolution et de Pooling. Voyons maintenant ces aspects un peu plus en détails.

Paramètres de la couche de convolution

Dans un réseau de neurones convolutifs, 3 paramètres principaux doivent être changés pour modifier le comportement d'une couche convolutive. Ces paramètres sont la taille du filtre, la foulée et le Padding.

1. Kernel size/Taille du filtre : une caractéristique de l'image peut être un bord vertical ou un arc, ou n'importe quelle forme dans l'image. Les filtres de petite taille collectent autant d'informations locales que possible, les filtres plus grands représentent des informations plus globales, de haut niveau et représentatives.

Dans notre cas, nous avons utilisé un filtre de petite taille car les images en entrée sont petites et nous pouvons les différentier grâce à de petits détails. Notons qu'en général nous utilisons des filtres avec des tailles impaires.

Choix : un filtre de taille 3x3.

2. Stride/Foulée : un filtre va se déplacer sur toute l'image avec une certaine foulée (correspond au nombre de pixels de déplacement du champ récepteur, évoluant en translation sur l'image) et calculer à chaque position les produits scalaires entre les entrées du filtre et l'entrée à la position i.

Dans notre cas, comme les images sont petites il ne fallait pas utiliser une stride trop grande.

Choix : une stride de 1.

3. Padding : le Padding est utilisé pour ajouter des colonnes et des rangées de zéros afin de maintenir les tailles spatiales constantes après la convolution. Cela peut améliorer les performances car cela permet de conserver les informations aux frontières.

Choix : un padding de taille 1.

3.2.3 Etape 3. Entraînement des données

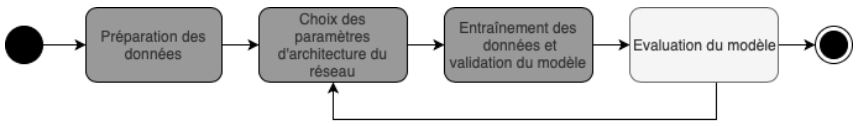

FIGURE 3.4 – Etape 3. Entraînement des données

Suite à la création du réseau de neurones, nous pouvons entraîner nos données en choisissant le nombre d'époques, la taille du batch et le learning rate.

Le nombre d'époques

Chaque fois que nous passons dans la boucle (appelée «époque»), nous calculons une passe en avant sur le réseau et mettons en œuvre une rétro-propagation pour ajuster les poids. Nous enregistrerons également d'autres mesures, telles que la perte et le temps écoulé, afin de pouvoir les analyser comme le réseau s'entraîne lui-même.

Il s'agissait ici de trouver un compromis entre le nombre d'époques, la taille du batch et le learning rate pour avoir les meilleurs résultats le plus rapidement possible.

Choix : 15 époques.

Taille du batch

On utilise un entraînement par batch qui consiste à rétropropager l'erreur de classification par groupes d'images. Cette méthode est plus rapide qu'en calculant l'erreur sur tout le jeu d'entraînement à chaque itération. Elle est plus stable, car les gradients d'erreurs ont moins de variance. À noter qu'un nombre trop important d'images par batch peut engendrer des problèmes de mémoire lors de l'exécution du code.

Choix : 40 images par batch.

Learning rate

Le taux d'apprentissage correspond à la vitesse d'apprentissage de notre réseau neuronal. C'est un hyper-paramètre qui contrôle à quel point nous ajustons les poids de notre réseau en fonction du gradient de perte.

Plus la valeur est basse, plus nous roulons lentement sur la pente descendante (et plus les temps de calculs sont longs). Si le taux d'apprentissage est faible, l'apprentissage est plus fiable, mais l'optimisation prendra beaucoup de temps.

Si le taux d'apprentissage est élevé, la formation peut ne pas converger. Les changements de poids peuvent être si importants que l'optimiseur dépasse le minimum requis et aggrave la perte.

Il existe plusieurs façons de sélectionner un bon point de départ pour le taux d'apprentissage. Une approche naïve consiste à essayer différentes valeurs et à déterminer laquelle donne la meilleure perte sans sacrifier la vitesse de l'entraînement. Nous pourrions commencer avec une valeur élevée telle que 0,1, puis essayer des valeurs plus faibles de manière exponentielle : 0,01, 0,001, etc.

Lorsque nous commençons à nous entraîner avec un taux d'apprentissage élevé, la perte ne s'améliore pas et augmente même lorsque nous exécutons les premières itérations de l'entraînement. Lorsque nous nous entraînons avec un rythme d'apprentissage plus faible, la valeur de la fonction de perte commence à diminuer à un moment donné au cours des premières itérations. Ce taux d'apprentissage est le maximum que nous puissions utiliser. Toute valeur supérieure ne laisse pas la formation converger.

Après plusieurs tests avec notre réseau 0.1, 0.01, 0.001, 0.0005 et 0.0001, le choix le plus pertinent semblait être 0.0005.

Choix : 0.0005.

3.2.4 Etape 4. Evaluation du modèle

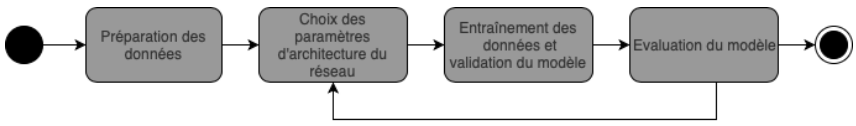

FIGURE 3.5 – Etape 4. Evaluation du modèle

Matrice de confusion

La matrice de confusion est, dans la terminologie de l'apprentissage supervisé, un outil servant à mesurer la qualité d'un système de classification.

Chaque colonne de la matrice représente le nombre d'occurrences d'une classe estimée, tandis que chaque ligne représente le nombre d'occurrences d'une classe réelle (ou de référence). Les occurrences utilisées pour chacune de ces 2 classes doivent être différentes.

Vous pourrez trouver des exemples de matrices de confusion dans la section suivante (Réponses aux questions subsidiaires).

Courbe ROC (Receiver Operating Characteristic,

La courbe ROC s'applique à un problème de classification binaire qu'on peut scinder en trois questions :

• Le modèle a bien classé un exemple dans la classe homologue.

• Le modèle a bien classé un exemple dans la classe non-homologue.

• Le modèle a bien classé un exemple, que ce soit dans la classe homologue ou la classe non-homologue. Supposons que nous avons un classifieur qui classe des observations en un ensemble de classes. De plus, il donne cette réponse accompagnée d'un score de pertinence. Deux cas sont possibles : soit la réponse est bonne (1), soit la réponse est fausse (0). Pour chaque observation, on associe un couple (r,x) où r est égal à 0 ou 1. x est le score de pertinence. On cherche à déterminer à partir de quel seuil de pertinence, la réponse du classifieur est fiable. En faisant varier x, on obtient une courbe.

Si les réponses sont liées, le modèle peut répondre de manière plus ou moins efficace à ces trois questions. On calcule les courbes ROC à ces trois questions.

Vous pourrez trouver des exemples de courbes ROC dans la section suivante (Réponses aux questions subsidiaires).

3.3 Réponses aux questions subsidiaires

3.3.1 Parmi les trois types de descripteurs, quel est le meilleur ?

J'ai choisi pour répondre à cette question de n'utiliser que les points GrayMax et de tester le modèle avec les différents descripteurs.

En appliquant les paramètres énoncés plus haut sur les données "eTPR_GrayMax - eTVIR_ACGT", nous obtenons :

On obtient une précision sur les données de test égale à : **96.4875 %**

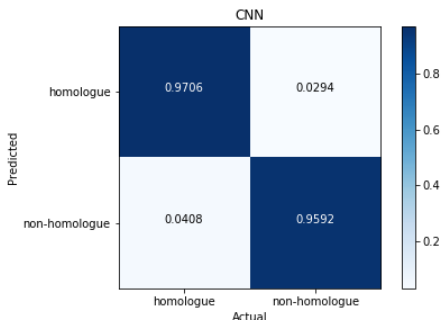

FIGURE 3.6 – Matrice de confusion obtenue pour GrayMax - ACGT

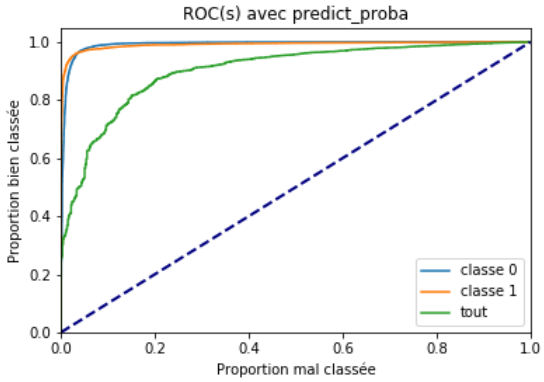

FIGURE 3.7 – Courbes ROC obtenues pour GrayMax - ACGT

En les appliquant à "eTPR_GrayMax - eTVIR_ACR0", nous obtenons :
On obtient une précision sur les données de test égale à : **94.2882 %**

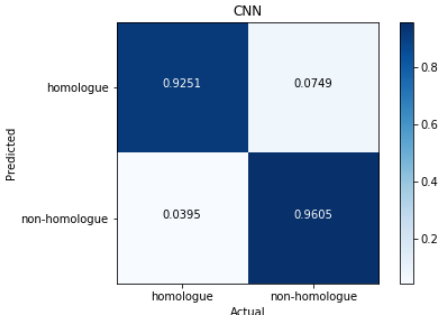

FIGURE 3.8 – Matrice de confusion obtenue pour GrayMax - ACR0

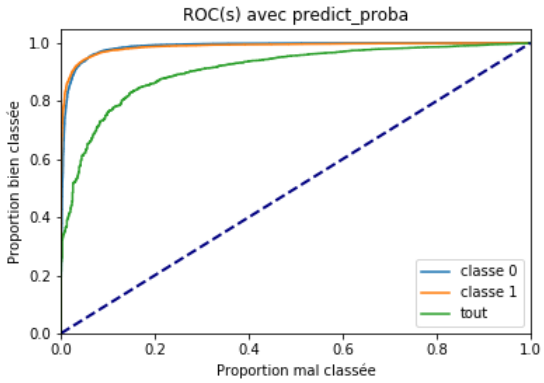

FIGURE 3.9 – Courbes ROC obtenues pour GrayMax - ACR0

En les appliquant à "eTPR_GrayMax - eTVIR_ACGR", nous obtenons :
On obtient une précision sur les données de test égale à : **94.8063 %**

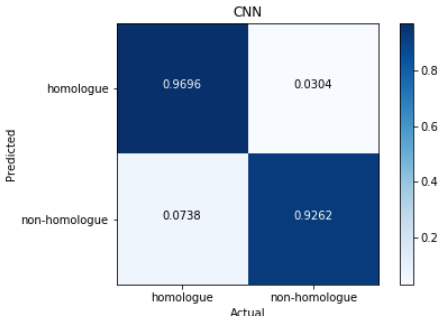

FIGURE 3.10 – Matrice de confusion obtenue pour GrayMax- ACGR

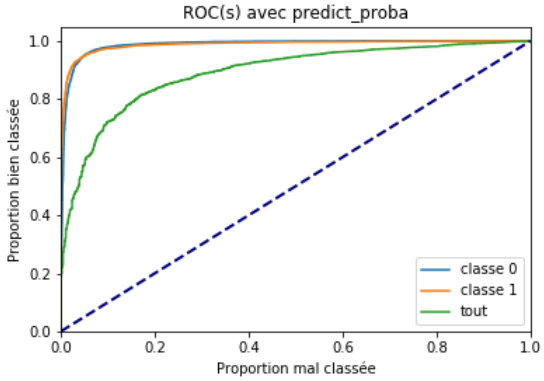

FIGURE 3.11 – Courbes ROC obtenues pour GrayMax - ACGR

Conclusion

Notre modèle semble mieux fonctionner sur le descripteur ACGT (corrélation de l'image avec le gradient tangentiel) qu'avec les autres descripteurs. On a en effet plus de chance de reconnaître la bonne classe pour ce descripteur (environ 2.2% de bonne reconnaissance en plus que pour ACR0 et 1.7% de plus que pour ACGR). Je ne fais ici le test que sur un seul type de point caractéristique, mais nous verrons par la suite que ACGT semble mieux fonctionner avec n'importe quel type de point. Cependant, les écarts ne sont pas très importants, l'utilisation des descripteurs ACR0 et ACGR reste possible et peut être efficace.

3.3.2 Parmi les différents types de points caractéristiques quels sont les meilleurs ?

Pour cela nous avons lancé l'apprentissage avec les paramètres évoqués plus haut sur tous les points caractéristiques, on a obtenu les résultats suivant :

	GrayMax	LaplMin	BifurqMin	GrayMin	LaplMax	BifurqMax
ACR0	94.29 %	95.25 %	94.23 %	94.30 %	94.68 %	94.60 %
ACGT	96.49 %	95.21 %	96.24 %	94.41 %	95.39 %	95.9 %

FIGURE 3.12 – Précision avec les différents points caractéristiques

Conclusion

D'après les résultats très proches les uns des autres, on ne peut pas déterminer qu'un type de point est meilleur qu'un autre. On remarque que la précision de GrayMax avec le descripteur ACGT est la plus élevée. Cependant, ces précisions varient quelque peu à chaque fois que l'on lance le programme, ce ne sont pas des résultats fixes. On ne peut donc pas affirmer qu'un type de point soit meilleur qu'un autre.

3.3.3 Est-ce que si l'on mélange les types de points, on gagne en score de précision ?

Pour cela nous avons utilisé tous les points caractéristiques (du même type de descripteur ACGT) pour réaliser l'apprentissage. J'ai récupéré 5 000 points homologues et 5 000 points non-homologues pour les 6 types de points caractéristiques et je les ai réunis dans un seul dossier (30 000 points homologues et 30 000 points non-homologues).

On obtient une précision sur les données de test égale à : **94.2833 %**

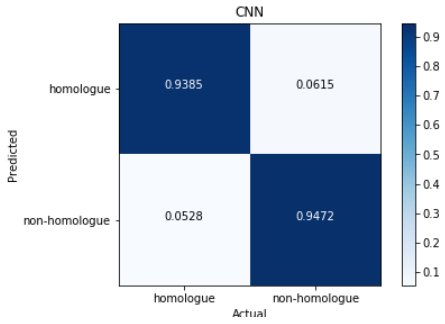

FIGURE 3.13 – Matrice de confusion obtenue pour tous les points caractéristiques avec le descripteur ACGT

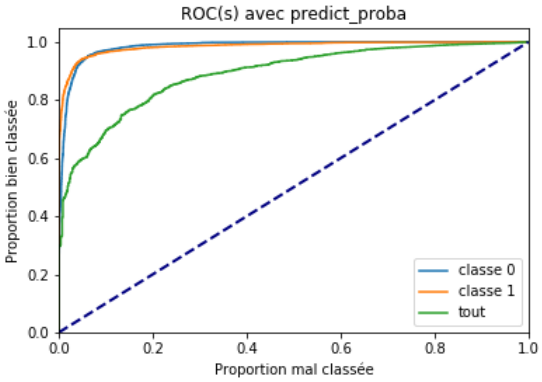

FIGURE 3.14 – Courbes ROC obtenues pour tous les points caractéristiques avec le descripteur ACGT

Conclusion

Le mélange des points caractéristiques ne semble pas améliorer les résultats de notre algorithme. Cependant, les résultats sont corrects, il est donc possible de mélanger les différents types de points pour l'apprentissage. Attention cependant à ne pas oublier de recalculer la moyenne et l'écart type pour la normalisation.

Chapitre 4
Problèmes rencontrés

Pendant cette phase de développement, j'ai rencontré des problèmes liés au temps de calculs. En effet, le volume des données à traiter est important. Il a fallu découper les images pour chaque type de point caractéristique et chaque descripteur puis lancer l'apprentissage sur toutes ces données séparément. De plus, Pytorch permet d'utiliser la puissance des GPU, or mon ordinateur n'en possède pas, j'ai donc simplement utilisé mon CPU.

Chapitre 5 Bilan et perspectives

5.1 Bilan

Au final, une architecture CNN a été créé et est effective. Un apprentissage sur différents descripteurs avec différents hyper-paramètres a été réalisé puis évalué grâce notamment aux matrices de confusion et aux courbes ROC. Pour mener à bien ce projet, j'ai passé beaucoup de temps à me documenter sur les réseaux de neurones et à comprendre sur quels paramètres jouer pour améliorer les résultats. J'ai pu en apprendre davantage sur le Deep Learning dans la théorie, ses applications en photogrammétrie et aussi sur les librairies utilisées en Python. Les codes sont par ailleurs disponibles sur GitHub au lien suivant https://github.com/GuillemetteF/CNN_points_homologues

Pour conclure sur les résultats, notre modèle obtient les meilleures précisions grâce aux points caractéristiques **GrayMax** et le descripteur **ACGT**.

On obtient une précision sur les données de test égale à : **96.4875 %**

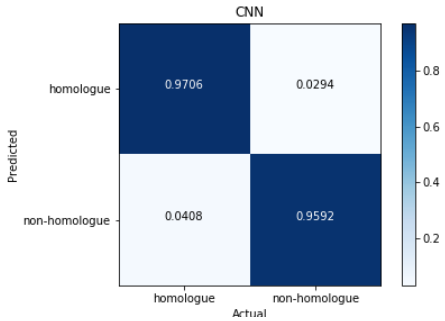

Figure 5.1 – Matrice de confusion obtenue pour GrayMax - ACGT

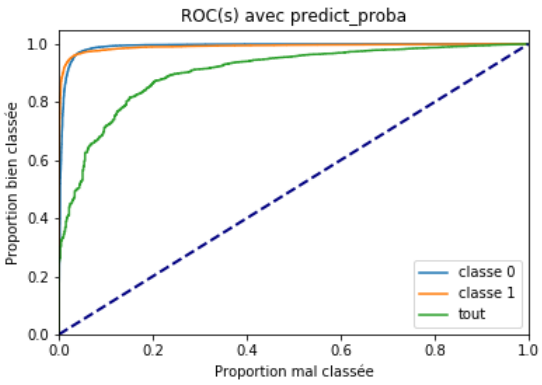

Figure 5.2 – Courbes ROC obtenues pour GrayMax - ACGT

5.2 Perspectives

On pourrait chercher à améliorer le score de l'algorithme.

5.2.1 Comment améliorer les performances ?

• Réalisation de réseaux siamois

On pourrait chercher à améliorer le résultat en utilisant un réseau siamois. Ce sont des réseaux neuronaux contenant deux composants de sous-réseau identiques. On pourrait donner l'image du descripteur d'un point à un réseau et l'image du descripteur homologue correspondant à l'autre réseau, les faire apprendre et assembler les images ensuite en indiquant qu'elles correspondent à des points homologues. Un réseau siamois peut ressembler à ceci :

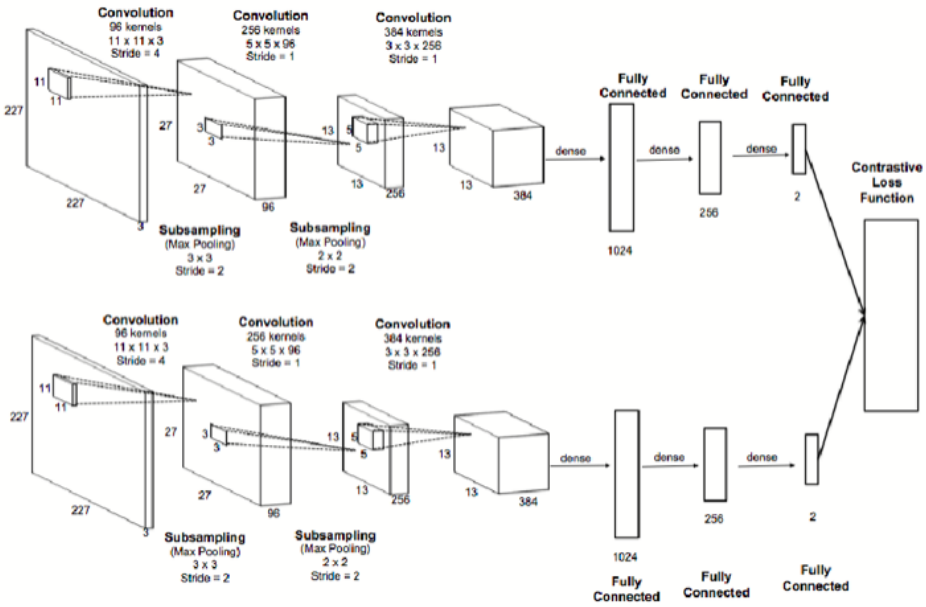

FIGURE 5.3 – Exemple d'un réseau siamois (source : Rao et al.)

• Changement de l'architecture du réseau neuronal et des hyper-paramètres

Il existe un nombre incalculable de combinaisons possibles pour créer un réseau de neurones. Dans notre cas, sachant que les images sont de petites tailles, il paraît évident que le nombre de couches de convolution et de Pooling seront restreintes. Cependant, toutes les architectures n'ont pas été testées. Je vous propose la meilleure que j'ai pu trouver, mais d'autres combinaisons pourront être envisagées.

Bibliographie

Généralités sur le Deep Learning :

- https://www.lebigdata.fr/deep-learning-definition
- http://penseeartificielle.fr/focus-reseau-neurones-artificiels-perceptron-multicouche/
- https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7
- https://deepsense.ai/keras-or-pytorch/
- https://rfiap2018.ign.fr/sites/default/files/ARTICLES/RFIAP_2018/RFIAP_2018_Gillot_Algorithmes.pdf
- Matrice de confusion
- Comment les Réseaux de neurones à convolution fonctionnent-ils?

- https://savoirs.usherbrooke.ca/bitstream/handle/11143/13397/Cornioley_Paul_MSc_2018.pdf?sequence=1&isAllowed=y
- Learning rate

Documentation des librairies :

- https://keras.io
- https://pytorch.org/docs/stable/index.html

Tutoriels utiles à la programmation :

Tutoriel Pytorch :

- https://github.com/pytorch/tutorials/blob/master/beginner_source/transfer_learning_tutorial.py
- https://pytorch.org/tutorials/

Tutoriel Keras :

- Apprenez à construire un CNN et gagnez du temps avec le Transfer Learning
- Image Classification in Python
- Tutorial: Optimizing Neural Networks using Keras
- Implémentation de perceptrons simples et multicouches

Aide à l'implémentation :

- Implémentation courbe ROC
- Choix des hyper-paramètres
- Enregistrer et charger un modèle
- Enregistrer et charger un modèle 2

Table des figures