Université Gustave Eiffel

ENSG Géomatique

ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES

Documentation

Technique et Méthodologique

ING22 - Mai 2024 **Version 1.0**

Sujet Geodev² : Détecter le monstre des marais dans les cartes anciennes de l'IGN

Lilia Campo | Suzanne Lizot | Elisa Peyrard | Vanessa Pech

Pour Hervé, Guillaume, Azelle et iamvdo...

Sommaire

I Contexte

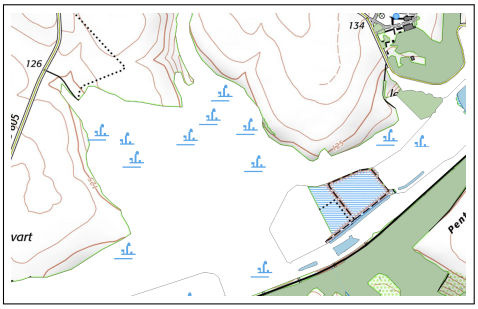
Le but de ce guide technique et méthodologique est de décrire la démarche et la méthode mise en oeuvre lors de la conception de l'application SwamPy implémentée dans le cadre des projets Géodev² 2024 de l'ENSG. L'application se trouve dans ce magnifique dépot git.

I.1 Rappel sur l'objectif de l'application

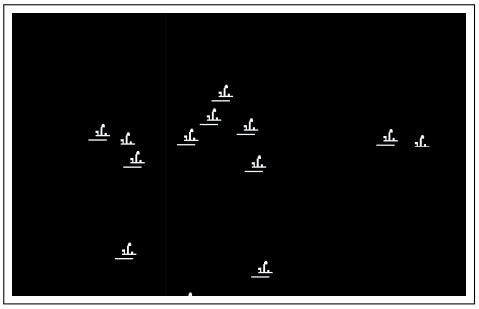
L'objectif principal de notre application est de détecter précisément les pictogrammes de marais sur des portions de cartes anciennes avec un algorithme de deep learning en donnant la possibilité de créer son propre jeu d'entrainement.

II Création du dataset

Le modèle d'apprentissage a besoin, pour s'entrainer de beaucoup de couples comme ci-dessous : une portion de carte qui se rapproche le plus d'une carte ancienne au niveau du style (image modèle)/les pictogrammes de marais de la carte en image binaire (masque associé)



(a) Carte créée avec des données actuelles qui ressemble à une carte ancienne



(b) Pictogrammes de la carte en image binaire

FIGURE 1 – Couple d'images modèle/masque

Spécifications des images

Les images générées pour créer le dataset sont des images :

- ❖ de taille 256x256 pixels
- ❖ de format PNG

II.1 Création de l'image modèle et du masque associé

Nous utilisons QGIS pour créer ces images.

L'image modèle

- ❖ Pour recréer le style d'une carte ancienne :

  - Allez dans l'onglet *Symbologie* de la couche vecteur et changez la couleur/-symbole de cette couche pour qu'elle ressemble le plus possible à la carte ancienne.

Vous pouvez utiliser l'outil pipette pour capturer le pigment exacte de la carte ancienne pour éviter de prendre les couleurs au hasard à vue d'oeil.

- Toujours dans *Symbologie*, chercher le bouton qui permet de télécharger le style de la couche dans un fichier qml.

❖ Pour charger des styles qml existant, vous devez utiliser la fonction *Appliquer le style* qui se trouve dans la boîte à outil de traitements.

Pour éviter de faire ça à la mano pour toutes les couches vous pouvez écrire dans la console python de QGIS. Une boucle for sur les couches du projet avec la fonction d'application de style qui va bien, plutôt simple si nos fichiers qml ont le même nom que les couches.

Le masque

Nous utilisons encore des manipulations QGIS pour créer ce masque. Toutes les fonctions utilisées sont trouvable dans a boîte à outil de traitements. Dans votre projet QGIS avec les couches vecteurs qui ont un style de carte ancienne :

1. Utiliser la fonction *Créer une grille* avec les paramètres suivant :

   ❖ Type de grille : Rectangle

   ❖ Etendue de la grille : Au choix, soit l'emprise de vos couches de marais, caneva de la carte ou dessiner l'emprise mais ne définissez pas une étendue trop grande. Une zone avec des pictogrammes avec pleins de contexte autour différent, pourquoi pas !

   ❖ Espacement horizontal et vertical : 2 km

   C'est la taille d'un carré de la grille, cela a été choisi de façon "empirique" et de sorte à ce que la génération de la grille ne soit pas trop longue et que la zone ne soit ni trop grande ni trop petite pour ne pas contenir seulement des pictogrammes.

2. Nous allons créer une couche qui correspond à l'intérieur des marais, ce sera dans cette couche que l'on va générer les pictos donc on évite d'en générer trop proche des bords qui dépasseraient de la couche de base. Utiliser la fonction *Tampon* avec les paramètres suivant :

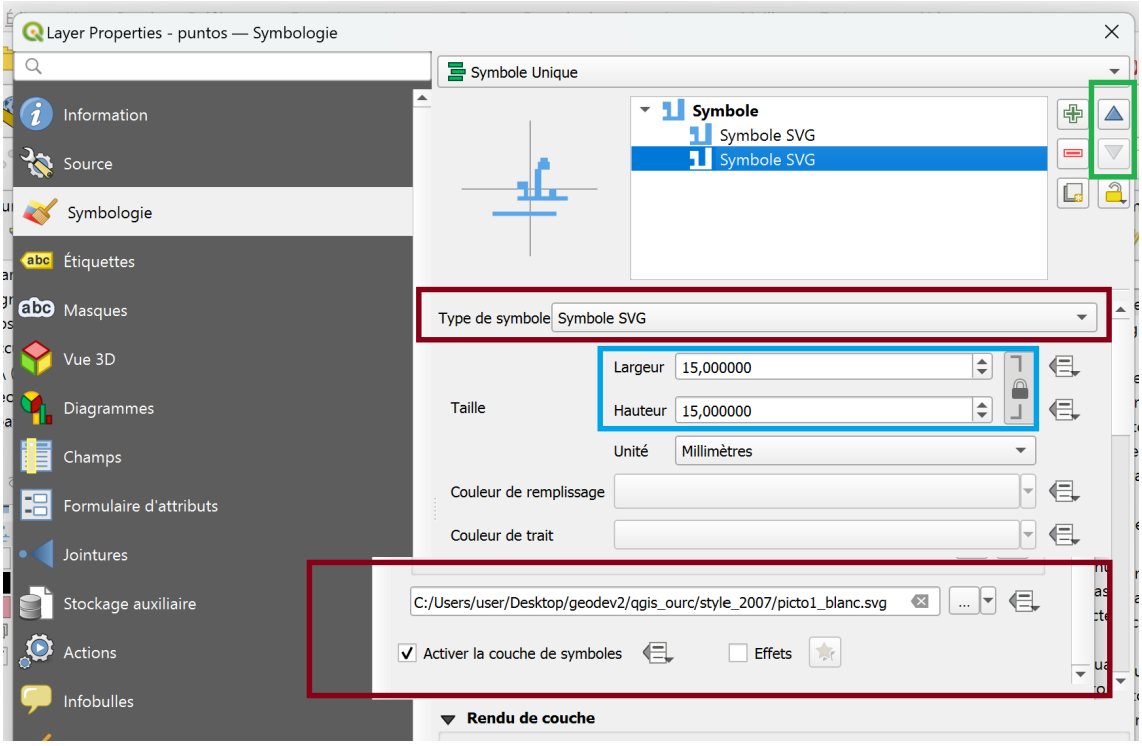
   ❖ Couche source : Votre couche de marais

- ❖ Distance : -50 m
- ❖ Vous pouvez laisser les autres paramètres par défaut et cochez Regrouper le résultat.

3. Pour générer les pictogrammes, utilisez la fonction *Points aléatoires à l'intérieur des polygones* :

   - ❖ Couche source : Votre couche de marais **intérieure**
   - ❖ Stratégie d'échantillonnage : Densité de points
   - ❖ Densité : 0,000010
   - ❖ Distance minimale entre les points : 100 m

4. Sur la couche des points, dans *Symbologie* :



- Les rectangles rouges sur l'image indiquent comment mettre le style SVG, mettez le chemin des pictogrammes SVG en bleu et en blanc

- Rectangle bleu, les dimensions du picto : 15 ou 20 pixels ou à vous de déterminer. Testez, générez le tiff et voyez si cela convient bien (cela dépend aussi de la taille du picto SVG c'est pour cela)

- Rectangle Vert : afficher le picto bleu d'abord ou le picto blanc

Le fait de mettre le symbole SVG des pictos bleus et blancs sur la même couche de points assure que les pictos bleus et blancs ont bien la même taille et se superposent, ce qui n'est pas forcément le cas si on duplique la couche de points (pour une raison que j'ignore). Lorsque vous voulez afficher le masque, mettez le picto blanc en premier.

Le code présent dans le git du projet model_mask_qgis.py représente ces manipulations QGIS en lignes de code python. Comme cela peut être assez fastidieux pour bien mettre les bons chemins de fichier à chaque fois car le code écrit comme tel vous demande de faire ce travail malheureusement, il est probable que cela ne fonctionne pas. Donc il serait plus simple et plus rapide de faire ces manipulations QGIS directement.

Le mieux pour automatiser la création du masque est de faire plutôt un plugin QGIS mais malheureusement nous n'en avons pas eu le temps.

II.2 Génération des tuiles

Lors de ce géodev, nous avons rencontré quelques imprévus, ce qui a fait que nous avons deux méthodes pour générer les tuiles. La première méthode n'a pas été présentée lors du projet mais présente l'avantage de garder les tuiles dans la même échelle et d'utiliser GDAL et non un code python aux origines douteuses (pardon je ne peux pas m'en empêcher).

Méthode 1

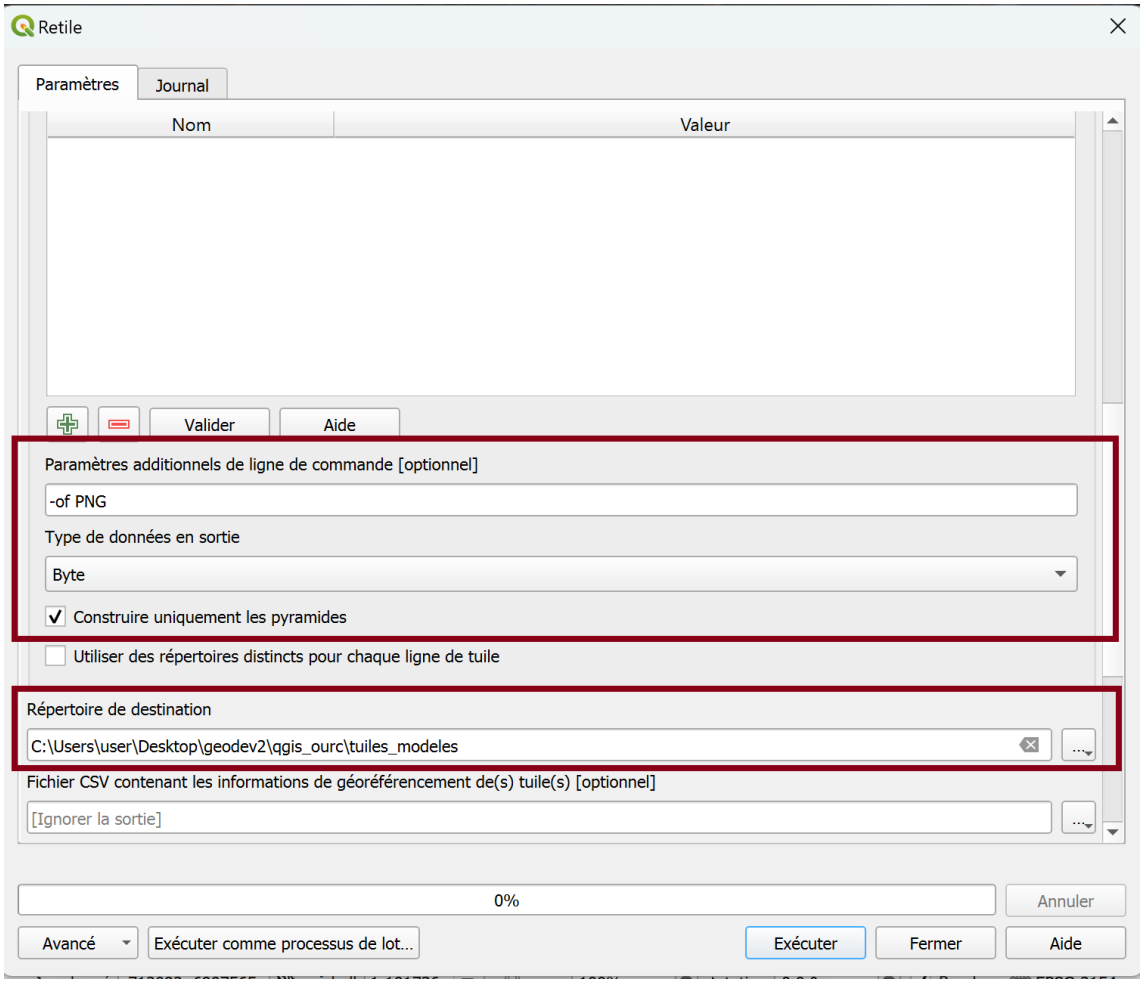
1. Créez le tiff de l'image à découper (le masque ou l'image modèle) avec la fonction *Convertir une carte en raster*

    - ❖ Etendue minimale pour le rendu : Au choix, calculer depuis la couche de la grille ou utiliser l'emprise actuelle du caneva de la carte. Avec cette deuxième option vous pouvez avoir plusieurs "point de vue" d'un même endroit donc plusieurs images et c'est moins prise de tête pour savoir si notre emprise n'est pas trop grande.

Ne pas choisir une emprise trop grande au risque de faire planter QGIS et bien évidemment garder la même emprise pour le masque et le modèle donc si vous utilisez le caneva de la carte ne vous déplacez pas avant d'avoir générer les deux images.

- ❖ Unité de carte par pixel : 1 (1m = 1px)
- ❖ Vous pouvez laisser les autres paramètres par défaut
- ❖ Enregistrer votre couche dans un répertoire

2. Vous avez enfin des rasters ! Bravo ! Nous allons pouvoir utiliser la fonction *GDAL Retile*, pour créer les tuiles en png



- Dans les paramètres avancés, mettez la commande *-of PNG*

- Changez le type en *Byte*

- Cocher la case *Construire uniquement les pyramides*

- Choisissez un répertoire de destination

- Laissez tous les autres paramètres par défaut (la taille de la tuile est par défaut à 256x256 px)

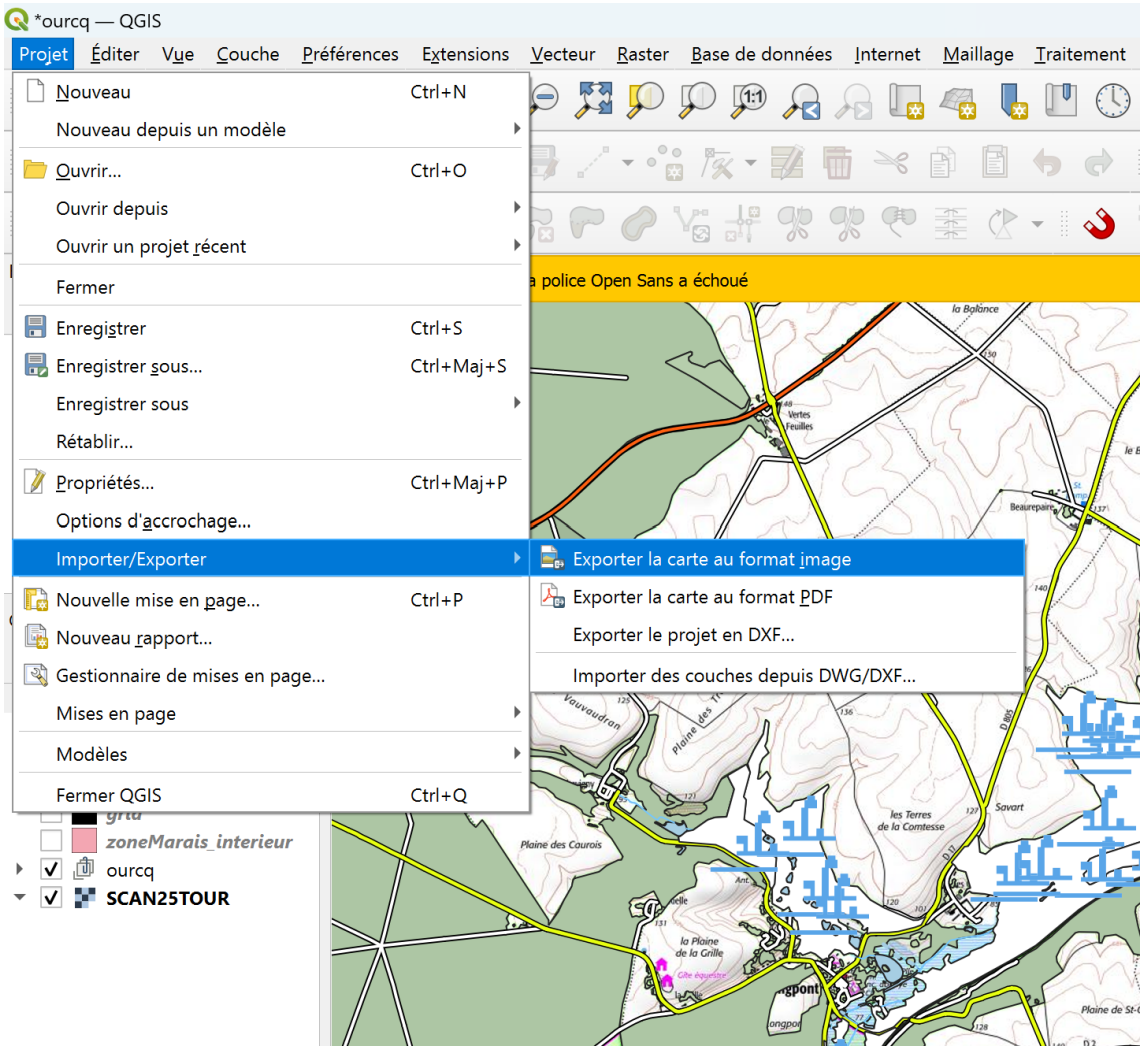
Les tuiles du masque générées par GDAL doivent être traitées, il faut les renommer en y ajoutant *"_mask"* dans leur nom et les transformer en binaire. Vous pouvez vous inspirer des fonctions python aux origines douteuses du git createImagette.py.

Vous pouvez utiliser GDAL aussi pour découper votre carte ancienne. Ouvrez votre fichier JP2 de carte ancienne et utilisez Retile and voilaaa !

Méthode 2

C'est donc la méthode qui se trouve dans le git.

1. Enregistrer vos masques et vos images modèles en exportant votre carte directement en PNG.



- Choisissez ensuite comme paramètre 300 dpi pour la résolution.

2. Mettez toutes vos images PNG générées précédemment dans le dossier modele si ce sont des images modèles sinon dans le dossier masque.

3. Lancez le script createImagette.py, attendez un peu et vous trouverez les tuiles dans les dossiers correspondant.

4. Créez un dossier Dataset avec deux sous-dossier : images pour y mettre vos tuiles du dossier tuile_modele et annotations pour y mettre vos tuiles du dossier tuile_masque.

Respectez bien la nomenclature du dossier de votre dataset ainsi que les noms de dossiers. Un dossier ***Dataset*** qui contient deux sous-dossiers ***images*** et ***annotations***.

Les deux méthodes présentent chacune leurs avantages et inconvénients. A vous de choisir celle que vous préférez.

III Modèle de Deep learning

III.1 Briques logicielles à utiliser

Pour pouvoir exécuter nos scripts python de deep learning :

Dans un environnement local

Il faudra que vous installer les librairies python de deep learning :

1. TensorFlow version 2.15 : Une bibliothèque open-source développée par Google. Elle est très populaire et offre une grande flexibilité pour la création et l'entraînement de modèles de Deep Learning.

2. Keras version 2.15 : Une interface haut niveau construite au-dessus de TensorFlow (et d'autres backends). Keras simplifie la création de modèles de Deep Learning.

```
pip install keras==2.15
pip install tensorflow==2.15
```

Dans un environnement Virtuel

Nous utilisons Google Colab, un environnement de notebook Jupyter basé sur le cloud. Il permet d'écrire et d'exécuter du code Python directement dans votre navigateur. Il est largement utilisé dans le domaine du Deep Learning, pour différentes raisons, notamment il vous donne un accès gratuit à des ressources informatiques puissantes telles que les unités de traitement graphique (GPU) et est livré avec des bibliothèques Python populaires telles que TensorFlow et Keras qui sont déjà installées.

Voici le lien pour accéder à notre google colab : lien du google colab

1. Copier le fichier sur votre drive pour pouvoir l'éditer comme bon vous semble.

2. Avant de l'exécuter, il faut que vous zippiez votre dossier Dataset et que vous le mettiez sur votre google drive.

III.2 Readme

Dans ce projet, le modèle de deep learning utilisé est le modèle U-net. Le modèle est construit de zéro, couche de neurones par neurones (cf la fonction unet du fichier *model_training.py*.

Voir le Readme du git qui fait office de document d'utilisateur pour cette partie.

III.3 Améliorations possibles

A la fin du projet, nous en avons déduit que les valeurs d'hyper-paramètres optimales seraient une taille de lot de 32 et 128 epochs mais nous aurions potentiellement obtenu de meilleurs résultats avec un nombre plus important d'epochs, avant d'atteindre le sur-apprentissage, mais cela n'a pas pu être vérifié pour cause de manque de temps.

Donc pour avoir un meilleur modèle, nous aurions pu aussi :

- ❖ Améliorer le dataset en augmentant le nombre d'images ou de la data augmentation, technique consiste à créer des versions modifiées des images existantes dans le dataset (ImageDataGenerator avec Keras)

- ❖ Notre modèle détecte des éléments qui ne correspondent pas à des pictogrammes de marais, comme le quaroyage bleu présent sur certaines cartes anciennes, ou des pictogrammes correspondant à une autre couche que celle

des marais. Il aurait fallu avoir dans le dataset des images avec ces éléments parasites.

- ❖ Nous avons fait varier deux valeurs d'hyper-paramètres mais il en existe d'autres, notamment nous aurions pu tester donc d'autres optimizer, un hyper-paramètres qui contrôle la mise à jour des poids du modèle pendant l'entraînement, nous avons laissé l'optimizer par défaut (cf la fonction unet de *model_training.py*)

- ❖ Utiliser un modèle U-net pré-entrainé et l'adapter à notre besoin de détecter des pictogrammes.

- ❖ Améliorer la ressemblance de nos tuiles de fausses cartes anciennes.

- ❖ Utiliser un autre modèle de deep learning comme Segformer.

En espérant que cette documentation vous a été utile et vous a donnée des idées pour vous occuper pendant votre temps libre, l'équipe LostInSwamp vous souhaite une très bonne continuation dans le monde du deep learning et des cartes anciennes.