Cycle des Ingénieurs de l'ENSG 3ème année
Spécialité PPMD

Rapport d'analyse final

Interface graphique d'aide à la création de réseaux de neurones

---

29 janvier 2024

Commanditaire : Arnaud LE-BRIS, IGN - LASTIG

Auteur : Malo DE LACOUR, ENSG

Sommaire

1. Contexte

Les méthodes par apprentissage profonds sont de plus en plus répandues pour réaliser diverses tâches (classification, régressions, ...). Différents outils tels que Pytorch, Tensorflow, Keras ... sont disponibles pour créer et entraîner ce type de modèle. L'une des étapes essentielles en apprentissage profond est la définition de l'architecture du réseau de neurones, c'est-à-dire définir l'enchaînement et la nature des différentes couches de neurones. Cette architecture est définie par du code. C'est l'un des facteurs sur lequel il est possible d'intervenir pour améliorer de manière significative les performances d'un modèle. De ce fait, cette architecture est amenée à être régulièrement modifiée avec parfois l'ajout d'étapes complexes. Cependant la diversité des enchaînements ainsi que des structures de réseaux rend parfois difficile la compréhension de l'architecture dans un environnement de développement classique.

De ce fait, une représentation schématique permettant de synthétiser et d'expliciter une architecture pour un humain s'avère pertinente. En effet, le concepteur pourra "dessiner" l'architecture du réseau de neurones et visualiser celle-ci de manière explicite pendant la phase de conception. Cet outil rendra la création de réseau de neurones intuitifs, ergonomiques et permettra de réaliser un gain de temps important.

2. Objectifs de l'étude

Le logiciel devra permettre à l'utilisateur de réaliser graphiquement son architecture et de le convertir sous forme de code python. L'essentiel du travail consistera à réaliser ou adapter une interface graphique visuellement approprié à la problématique des CNN.

2.1 Existant

2.1.1 Programmation par blocs :

Plusieurs domaines tels que le jeu vidéo (Blueprint dans Unreal Engine), la modélisation procédurale (Blender, Houdini, Speedtree) ainsi que les SIG (modelBuilder de ArcGIS) utilisent déjà la programmation par blocs. Ces interfaces de programmations visuelles ont surtout été développées pour rendre accessible le développement et la programmation (souvent complexe à appréhender) à des non-programmeurs. Cette solution ergonomique et intuitive permet entre autres à des designers et créateurs dans le milieu du jeu vidéo et des effets spéciaux de programmer des éléments. Cela représente aussi de nombreux avantages comme en SIG avec un gain de temps pour le développement et le déploiement de petits outils personnalisés.

2.1.2 Représentation visuelle des CNN

Il existe de nombreux outils tels que Visual Keras, neutron et conx qui permettent de visualiser la structure d'un réseau de neurones déjà implémenté. Ils permettent d'illustrer un réseau en montrant l'enchaînement des différentes couches ainsi que leur nature et dimension. Ces outils produisent un résultat très esthétique pour illustrer ou analyser mais ne permettent aucune interaction et ne sont donc pas adapter à la phase de conception. De

plus, certaines méthodes de visualisation ne sont pas adaptées à des architectures complexes.

Il existe cependant ENNUI et DeepLearning Studio qui sont des solutions très pertinentes pour répondre aux besoins de simplicité et d'ergonomie. Le problème de la première solution ENNUI réside dans le manque d'ergonomie pour manipuler des structures plus complexes. La seconde solution est propriétaire.

2.2 Contraintes

Aucune contrainte technique n'a été imposée par le commanditaire. Le choix de la librairie d'apprentissage privilégiée est Pytorch, cependant une librairie Tensorflow ou Keras convient très bien aussi. Il faut cependant intégrer un certain nombre de fonctionnalité décrite ci-dessous par ordre de priorité :

- **Représentation graphique** simple, interactive et ergonomique de l'architecture CNN
- Implémentation des couches et fonctions couramment utilisées en CNN
    - **Couches** : Pooling (max-min-avg) – Convolution – DenseLayer – Dropout – Batch-Normalisation – Flatten – Relu – Linear
    - **Opérations spécifiques** : Concaténation – Addition – Multiplication
      Permettre de convertir l'architecture représentée dans le logiciel en code python.
- **Permettre de sauvegarder** le travail réalisé sur l'interface dans un fichier spécifique pour reprendre ultérieurement.
- **Intégrer un système de contrôle** des flux (taille des données cohérentes d'un bloc à l'autre)
- Permettre de créer de nouveaux blocs personnalisés réutilisables à partir d'un ensemble de bloc de base

Une contrainte pourrait provenir de l'utilisation d'un code déjà existant. En effet, la création d'une interface graphique est très chronophage et parfois technique. C'est pourquoi il est très probable que je réutilise un code de programmation visuelle ce qui implique de s'approprier celui-ci. De plus, la structure du projet doit permettre d'ajouter de nouveaux blocs pour garantir l'évolutivité et la pérennité de celui-ci.

2.3 Objectifs

L'utilisateur doit pouvoir "dessiner" en toute simplicité son architecture. Il doit pouvoir réaliser des actions simples pour y parvenir. Ces actions sont représentées et synthétisées dans le diagramme de cas d'utilisation ***figure 1***.

- Ajouter et paramétrer un bloc
- Créer ou supprimer des connexions entre les blocs
- Convertir et enregistrer le schéma en code python
- Enregistrer le schéma pour conserver une version éditable

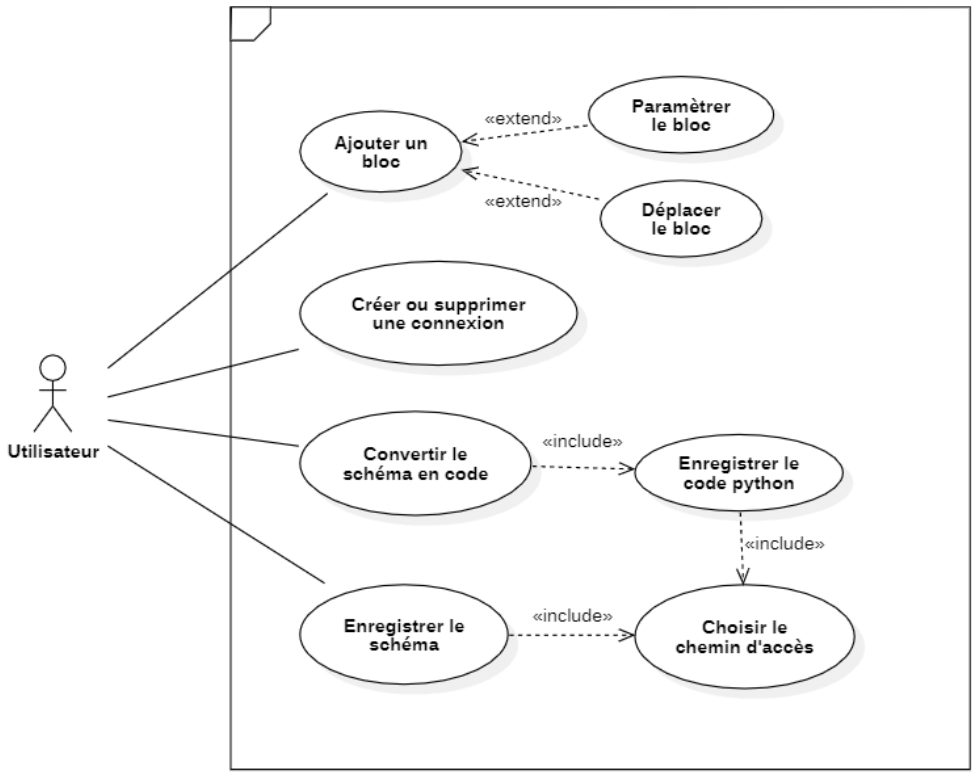

Figure 1 : Diagramme de cas d'utilisation

3. Analyse fonctionnelle

3.1 Architecture globale

Cette interface est un outil indépendant d'aide à la conception qui doit pouvoir fonctionner indépendamment de d'autres outils. Son architecture et son fonctionnement sont à réaliser de zéro. La solution de visualisation retenue est un système par bloc avec des connexions en entrée et sortie. L'utilisateur peut déplacer et connecter ces blocs avec un système de glisser/déposer. Une surbrillance des blocs et connexions permet à l'utilisateur de visualiser les éventuelles erreurs dans son architecture. Pour des questions d'ergonomie, la taille des données n'est pas directement visible dans le schéma de bloc.

Cette méthode de programmation nécessite de manipuler différents objets de base. Cela rend une structure orientée objet particulièrement pertinente pour la réalisation et l'évolutivité de ce projet. L'utilisateur est amené à manipuler et paramétrer ces objets :

- **un bloc (ou node)** : Cet objet correspond à une fonction avec ces paramètres ou à une variable. Il est constitué d'entrées et sorties matérialisé par des pins.
- **des paramètres** : Ce sont différents types d'objets qui peuvent composer un bloc tel que des champs numériques, boutons et curseurs. Ils permettent de saisir des paramètres comme des valeurs ou des méthodes que le bloc utilise.

- **une accroche (ou pin)** : C'est un morceau de bloc sur lequel vient s'accrocher une connexion. Chaque pin correspond à un paramètre et matérialise une entrée ou sortie de bloc
- **une connexion** : Elle permet de relier les blocs entre eux en s'accrochant aux pin. Elle est définie par un début et une fin. La connexion est représentée par un "cable" pour matérialiser le lien entre les blocs.
- **un schéma** : Le schéma est l'ensemble de blocs et connexion avec les paramètres qui y ont été renseigné. Il représente l'architecture du réseau en entier.

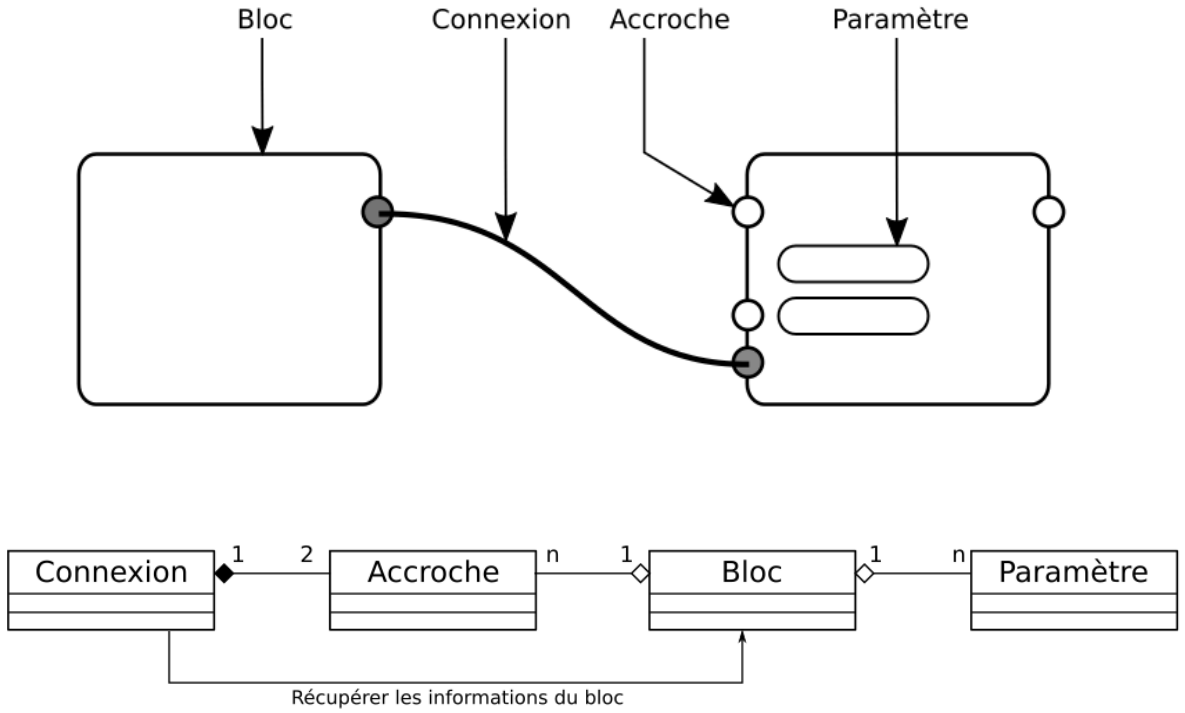

Figure 2 : Diagramme de classe simplifié des objets et leur représentations graphique

La **figure 2** représente les liens entre les différents objets décrits précédemment à l'aide d'un diagramme de classe.

3.2 Utilisation du logiciel

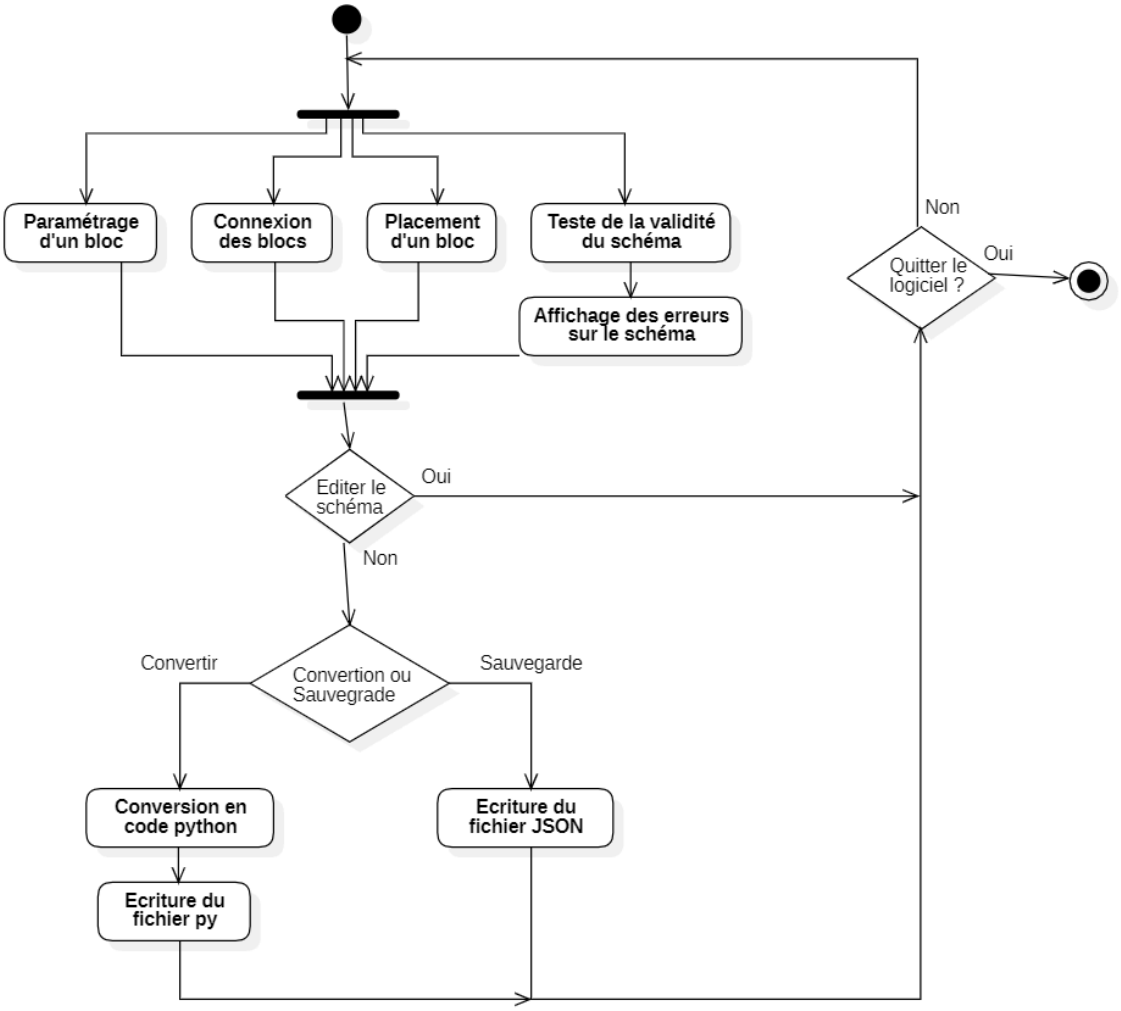

Figure 3 : Diagramme d'activité de l'interface

La **figure 3** montre la suite d'actions nécessaires pour réaliser un code python et une sauvegarde de schéma. La première phase consiste à éditer le schéma à l'aide de différentes actions (connexion, création de bloc, etc...). Un contrôle de validité est réalisé en parallèle de l'édition pour indiquer à l'utilisateur d'éventuelles erreurs à corriger.

Dans un deuxième temps, il y a une phase de conversion/sauvegarde pour permettre à l'utilisateur de sauvegarder une version éditable ou convertir en code python son schéma. Cette étape constitue une sortie temporaire de la phase d'édition, mais l'utilisateur revient en édition juste après.

Il est possible de quitter le logiciel à tout moment et donc de mettre fin au processus sans nécessairement réaliser d'action.

4. Réalisation technique

4.1 Architectures détaillées

4.1.1 Architecture initiale

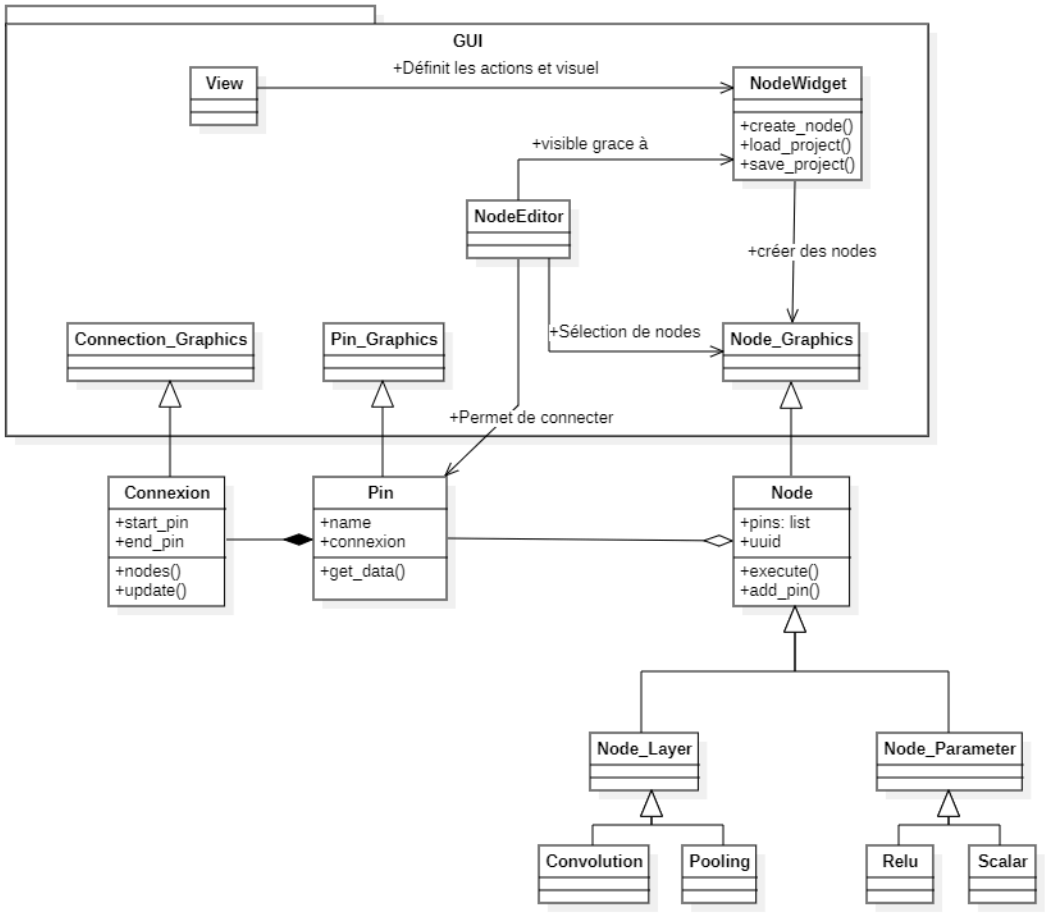

Figure 4 : Diagramme de classe initial détaillé avec les classes déjà implémenté par la librairie

La **figure 4** montre une architecture détaillée de l'interface en se basant sur l'architecture élaborée par la librairie. Ainsi, le bloc GUI constitue les éléments de la fenêtre, l'espace d'édition et les fonctions d'interaction avec celle-ci (Sélection, drag and drop, création de nouveaux objets). Ce bloc est légèrement modifié par la suite. Les classes *Connexion*, *Pin* et *Node* sont à compléter avec des méthodes pour les faire communiquer et exécuter des actions. Toutes les fonctions et variables matérialisé par des blocs sont des classes qui hérite de la classe *Node,* cela permet une implémentation simple de toutes les couches utiles dans une architecture.

4.1.2 Evolution de l'architecture

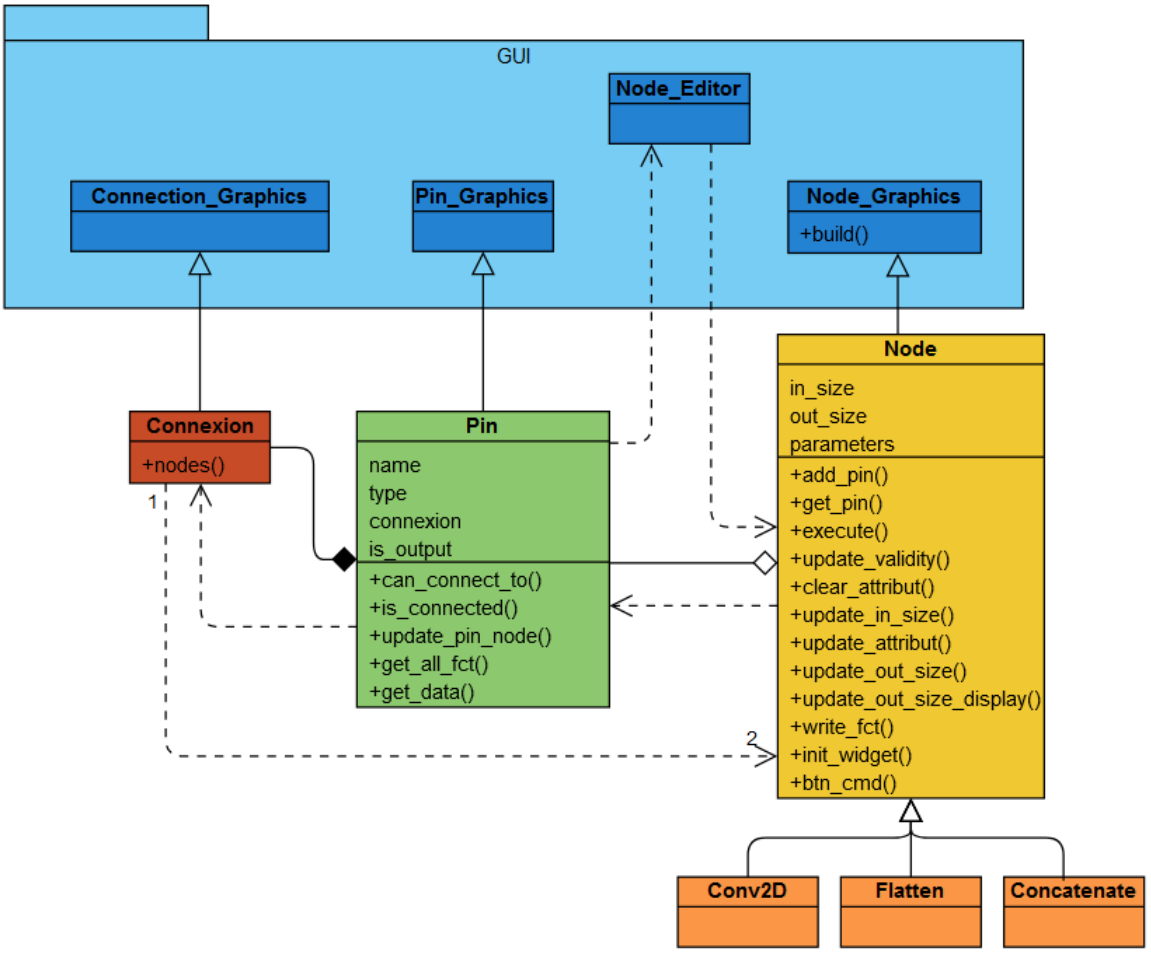

Figure 5 : Diagramme de classe final

La **figure 5** représente le diagramme de classe du projet réalisé. La structure globale n'a pas vraiment évolué par rapport à la structure décrite lors de la phase d'analyse. Les liens entre les différentes classes notamment entre Pin, Connexion et Node sont précisés avec les méthodes utilisées par chacun.

4.2 Fonctionnalitées implémentées

Fonctionnalité	Réalisation	Difficulté	Temps
Représentation graphique	Oui	/	/
Couches et opérations	Partielle	Faible	Important
Conversion du schéma	Oui	/	/
Sauvegarde	Non	Moyenne	Moyen
Contrôle et aperçu des flux	Oui	/	/
Factorisation de blocs	Non	Importante	Important

Figure 6 : Tableau des fonctionnalitées finales

L'intégralité des fonctionnalités initialement formulée par le commanditaire n'ont pas été implémentés. La **figure 6** récapitule l'ensemble des fonctionnalités avec leur état d'implémentation. Seule la factorisation de blocs, qui consiste à regrouper un ensemble de blocs élémentaire en un seul, constitue un véritable obstacle technique pour la finalisation de l'interface. En effet, cela implique de reconstruire l'affichage graphique du bloc ainsi que son fonctionnement.

4.3 Description du fonctionnement

4.3.1 Construction d'un bloc

Un bloc représente une fonction avec ces paramètres en entrées et sorties. Il possède des attributs graphiques (nom, type, couleur …) nécessaires à la visualisation et des attributs correspondants aux entrées et sorties de paramètres. Il est construit avec l'instanciation de pins pour connecter ses paramètres à d'autres blocs.
Un bloc classique comporte les méthodes décrites ci-dessous, les blocs de paramètres d'entrées et de sortie ne contiennent pas nécessairement toutes ces méthodes.

- ***update_validity*** : Regarde les paramètres et détermine la validité du bloc. Change la couleur du rectangle de validité selon le résultat. retourne un booléen de validité.
- ***clear_attribut*** : Réinitialise la valeur des attributs en paramètre.
- ***update_in_size*** : Enregistre les valeurs des données en entrée.
- ***update_attribut*** : Enregistre les valeurs des paramètres en entrée.
- ***update_out_size*** : Calcul et enregistre les valeurs des données en sortie.
- ***update_out_size_display*** : Actualise l'affichage de la taille des données en sortie.
- ***write_fct*** : renvoie la ligne de commande python à écrire.
- ***init_widget*** : permet d'ajouter des widgets (bouton, ligne d'édition, liste d'options …).
- ***build*** : construit la représentation graphique du bloc
- ***execute*** : retourne la liste des blocs dans l'ordre de connexion
- ***btn_cmd*** : Exécute l'ensemble des méthodes pour actualiser le bloc, ces attributs et la taille des données en entrée et sortie.

4.3.2 Types de pins

Un pin (aussi appelé accroche) est défini par un nom, un état (entrée ou sortie) et un type de donnée. Le type de donnée correspond au type de la variable qui doit transiter par ce pin. Les types de données utilisés sont décrits dans le tableau en **annexe A**. Ce type est représenté par une couleur. Par exemple, un pin de nombre entier est vert. Il ne peut etre relier qu'a un pin de même type, c'est à dire de couleur verte. Ainsi, cette contrainte de connexion permet à l'utilisateur de visualiser facilement le type de donnée en paramètre et d'éviter des connexions fausses qui provoqueraient des erreurs.

4.3.3 Transfert de données entres blocs

L'objet Pin a une méthode ***get_data*** pour récupérer l'information du bloc précédent. Pour cela, le pin prend la connexion à laquelle il est associé. Cette connexion possède une méthode ***nodes*** pour récupérer les 2 blocs qu'il connecte. Ainsi la méthode ***get_data*** retrouve l'attribut de paramètre du bloc connecté.

4.3.4 Exécution du schéma

L'exécution du schéma consiste à récupérer l'ensemble des blocs dans l'ordre afin de générer le code python associé à ce schéma. Pour cela, l'objet Pin possède la méthode ***get_all_fct***. Cette méthode est exécutée par récurrence, elle crée une liste et y ajoute le bloc auquel elle appartient. Si ce bloc est connecté à un bloc précédent avec une connexion de type data, alors la méthode est ré-exécuté sur le bloc précédent pour obtenir la liste correspondant à ce bloc. La liste obtenue par récurrence est une imbrication de liste qui doit être désimbriquée. Enfin, la méthode ***write_fct*** est appliquée à tous les blocs contenus dans cette liste pour générer le code python final.

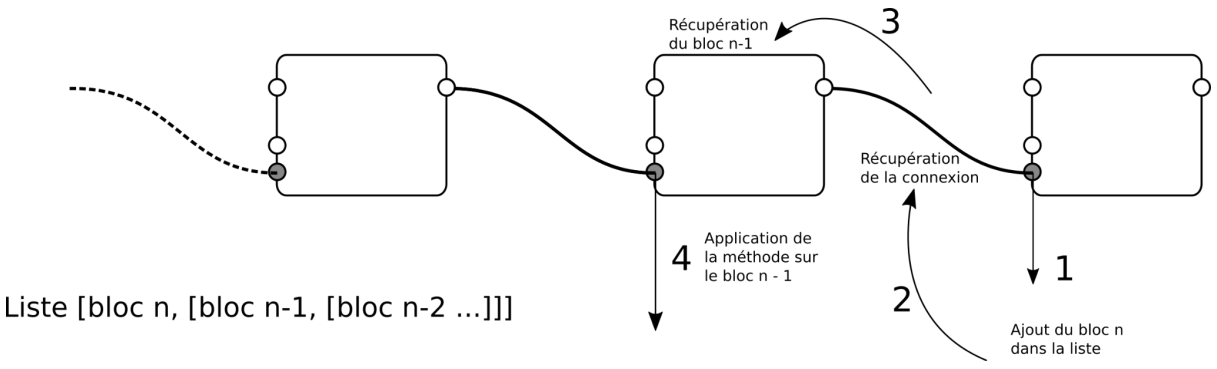

Figure 7 : Application de la méthode get_all_fct par récurrence

La **figure 7** décrit le processus récursif utilisé pour récupérer la liste de bloc dans l'ordre.

5. Utilisation et améliorations

5.1 Défauts actuels

Lors de l'utilisation de cette interface, les paramètres des blocs ne sont pas directement visibles. De plus, il n'est pas possible de saisir une valeur directement dans un bloc de type fonction, il faut obligatoirement utiliser un bloc de type input pour modifier un paramètre.

Un bloc est actualisé avec les valeurs en entrée uniquement lorsqu'il est connecté ou déconnecté. Cela implique un problème majeur. Si une valeur est modifiée dans un bloc déjà connecté, le bloc suivant ne sera pas actualisé et ne prendra pas ce changement en compte. Ainsi, une modification en début de schéma ne sera pas répercutée sur la suite. Ce problème peut facilement être contourné en actualisant les blocs dans l'ordre à partir de la liste de blocs générés lors de l'exécution du schéma. Cependant, cela représente une perte d'ergonomie importante.

5.2 Améliorations possibles

5.2.1 Script exemple pour l'implémentation de nouvelles fonctions

L'utilisateur peut être amené à utiliser des couches qui ne sont pas implémentées dans l'interface. C'est pour cela que ce projet contient un script permettant de comprendre comment créer un bloc complètement nouveau. Ce script exemple ***Example_node.txt*** reprend les éléments essentiels pour créer un nouveau bloc. Il donne la structure et un certain nombre d'explications sur la manière de construire une nouvelle fonction.

5.2.2 Fonctionnalités supplémentaires

Un certain nombres de fonctionnalités n'ont pas été formulé par le commanditaire et seraient très intéressantes pour gagner en ergonomie et en flexibilité.

- **Menu d'option d'un bloc :** Ce menu temporaire apparaît sur le côté de la zone d'édition du schéma lors de la sélection d'un bloc. Il permet de visualiser et d'éditer l'ensemble des paramètres du bloc.
- **Multi connexion des blocs:** Pour le moment, une sortie ne peut être connectée qu'à une seule entrée. La multi connexion permet de connecter une sortie avec plusieurs entrées pour factoriser les blocs paramètres en ré-utilisant la même valeur à plusieurs endroits.
- **Choix de la conversion :** Lors de l'exécution du schéma, une méthode ***write_fct()*** permet d'écrire la ligne de code correspondant au bloc en question. L'interface convertit le schéma en modèle Keras. Il est possible d'ajouter des méthodes pour réécrire le modèle en version Pytorch. Ainsi l'utilisateur peut simplement choisir le type de modèle qu'il souhaite (Keras, Pytorch ou autre) au moment de l'exécution du schéma.

6. Gestion de projet

6.1 Les risques

Intitulé du risque	Type	Impact	Probabilité	Solutions
Prise en main d'une librairie d'apprentissage	technique	majeur	moyenne	- Bonne documentation
Manque de temps	humain	majeur	moyenne	- Bonne organisation, - réduction des fonctionnalités
Prise en main du code interface	technique	majeur	faible	- Bonne documentation
Modification ou ajout de fonctionnalités	humain et technique	mineur	faible	- Echanges réguliers avec le commanditaire

Figure 8 : Tableau des risques initiaux

La **figure 8** représente les risques identifiés durant la phase d'analyse. La prise en main du code a en effet nécessité plus de temps que prévu. Le manque de temps est le principal facteur qui a impacté le projet. La prise en main du module pyQt est aussi un risque qui c'est produit mais qui n'avait pas été clairement identifié au départ.

6.2 Planning prévisionnel

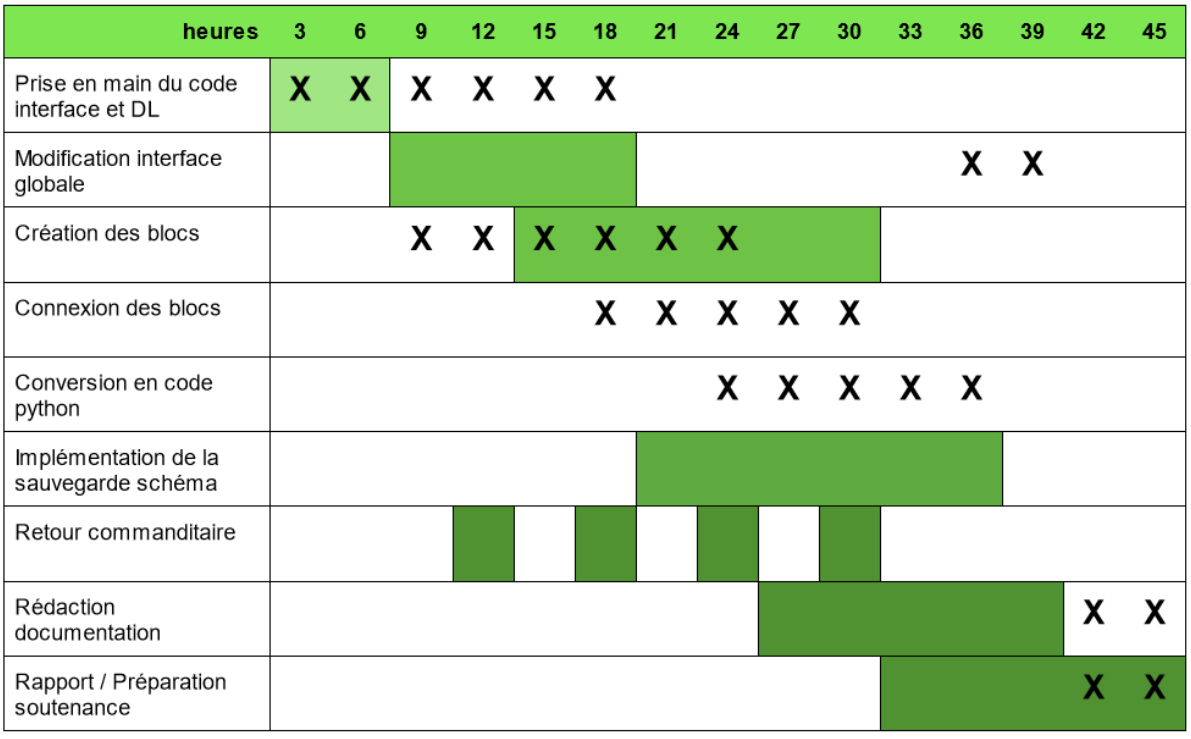

Figure 9 : Diagramme de Gantt de la phase de développement

La **figure 9** représente l'organisation et la répartition des principales tâches dans le temps envisagé au début ainsi que la répartition réelle du travail effectué. La prise en main du code d'interface à nécessité plus de temps que prévu. L'une des tâches principales qui n'avait pas été identifiées au départ à été de réaliser la connexion entre les blocs. La conversion du schéma en bloc est aussi une tâche importante qu'il aurait fallu préciser.

7. Synthèse

Ce rapport donne un aperçu du travail réalisé et de la manière dont je m'y suis pris ainsi que des résultats obtenus. La conception orientée objet de ce projet à été à la fois source de difficulté mais aussi source de réflexion pour réaliser un code facilement réutilisable et améliorable par la suite. La structure du code utilisé à permis une grande clarté malgré les difficultés pour prendre en main le module pyQt.

Annexes A

Type de paramètres	Nom	Type python	Couleur
Données d'entraînement	data	list	blanc
Entier	integer	int	vert
Ratio	rate	float	jaune
Noyau 2D	list2	list	bleu
Noyau 3D	list3	list	bleu foncé
Méthode d’activation	activation	str	orange

Tableau récapitulatif des types de connexions et de leur couleur