Cégep du Vieux-Montréal

Projet synthèse

Réalisé par

Afsaneh Sheikhmiri

Présenté à

Jean-Christophe Demers

Dans le cadre du cours

Projet synthèse
410-B65

24 février 2020

Table des matières

Présentation générale du projet

Mon projet synthèse aura pour but de créer une application qui permet de générer des images de tumeurs cancéreuses artificielles afin de pouvoir avoir un grand *data set* d'images. Avec ce *data set,* la machine pourra s'entrainer à apprendre à identifier les tumeurs sur les images générées et déterminer à quel stade de cancer elles sont atteintes.

Présentation précise du projet

Comme mentionné au début du document, mon projet aura pour but de générer des images de tumeurs cancéreuses artificielles afin de permettre à la machine de s'entraîner à identifier les tumeurs et interpréter leur stade.

Tout d'abord, ce projet sera réalisé en *Python*. Une des raisons pour lesquelles j'ai décidé d'utiliser ce langage est pour les modules qu'il offre. Plusieurs modules de *Python* me permettront d'accomplir des tâches avec plus de facilité. Plusieurs de ces modules implémentent des algorithmes pour l'intelligence artificielle et le *Machine Learning*. Le langage est puissant et l'implémentation des algorithmes sont plus pratique. Aussi, s'il y a des changements durant le projet, il sera plus facile d'expérimenter rapidement de nouvelles idées et des prototypes de code dans un langage avec une syntaxe minimale comme *Python*.

Ensuite, mon application devra être capable de détecter 5 stades. Les tumeurs de stade 2 auront la plus petite forme de mon application. Les tumeurs de stade 1 sont très petite, alors j'ai décidé de ne pas l'impliquer dans mon application. Les tumeurs de stade 3 et 4 sont des tumeurs plus grandes. À partir des recherches faites sur internet, je devrais décider les paramètres

de chaque tumeur d'un stade spécifique. Les tumeurs de stade 5 sont des tumeurs non-cancéreuses. Or, je voudrais avoir la possibilité de générer des images avec des tumeurs non-cancéreuses et permettre à l'application de pouvoir différencier entre une tumeur cancéreuse et non-cancéreuse. Le dernier stade correspond aux images, qui n'ont aucune tumeur. Donc, mon application pourrait être capable de détecter s'il y a une tumeur ou non.

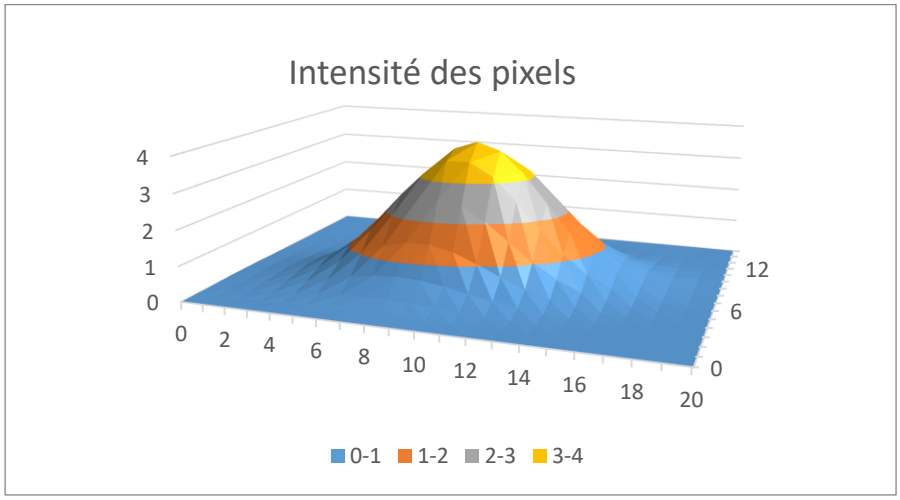
Aussi, mon application permet à l'usager de demander le nombre d'images qu'il veut générer en lui spécifiant qu'on aura besoin d'au moins 200 images. La raison pour laquelle je demande un minimum d'images est qu'il faudra un assez grand *data set* pour que l'application puisse pratiquer et apprendre à différencier les tumeurs.

Pour commencer l'application, l'usager doit peser sur le bouton « Prepare » afin de pouvoir générer les images. Pendant la création des images, il y aura une barre de progression pour montrer à l'usager où l'application est rendue et quand celle-ci va finir de générer les images. L'usager ne pourra pas voir les images qui ont été générées sur l'application. Il devra aller voir le dossier créé pour ces images afin de tous les visualiser. Pour pouvoir générer des images, je vais utiliser le module *Mathplotlib* de *Python* en raison de sa facilité à pouvoir manipuler une image. Ce module me permet de convertir mon tableau 2D en image de format **PNG**. La raison pour laquelle je veux convertir en **PNG** est que la compression est sans perte, ce qui signifie qu'il n'y a pas de perte de qualité à chaque fois qu'elle est ouverte et enregistrée à nouveau. Chaque image créée contiendra ou non une tumeur. Il faut savoir que la tumeur créée ne ressemblera pas exactement à ce qu'une tumeur cancéreuse ressemble en vrai sous un microscope. Dans ce projet, une tumeur aura la forme d'une ellipse. Puisque le module *Mathplotlib* peut convertir un tableau 2D en une image, nos objets de la classe *tumor_image* auront comme attribut une liste 2D. Chaque élément de la liste sera un pixel. Mon but est d'aller modifier chaque pixel afin de créer une ellipse.

Pour pouvoir changer les couleurs de chaque pixel, j'utilise la fonction gaussienne. À chaque pixel, j'applique la fonction gaussienne afin de le modifier. La fonction est celle –ci :

$$f(x,y) = Ae^{-(a(x-\mu_x)^2+2b(x-\mu_x)(y-\mu_y)+c(y-\mu_y)^2)}$$

Où « A » représente l'amplitude de notre distribution, le coefficient « a » représente la hauteur du pic de la courbe, le coefficient « b » représente la position du centre du pic et le coefficient « c » représente la largeur de la courbe. (μ_x, μ_y) représente le centre de notre courbe gaussienne. Cette fonction détermine l'intensité de chaque pixel. Or, on essaye alors de créer une sorte de distribution normale sur l'image, le plus proche qu'un pixel est du centre de la courbe gaussienne, plus son intensité est élevée et plus il apparait plus clair dans l'image. Dans le sens contraire, si un pixel est éloigné du centre de la courbe, celle-ci est beaucoup moins intense. Une courbe gaussienne ressemble au diagramme ci-dessous :



Le diagramme représente une image de 12 pixels en largeur et 20 pixels de hauteur. Le plus qu'on se rapproche du centre, le plus nos pixels augmente en intensité (plus la courbe monte). Or, dans l'image généré le centre devrait être le pixel le plus illuminé.

Aussi, pour chaque pixel, je vais utiliser le module noise afin de donner un effet plus réaliste dans l'image générée.

Peser sur le bouton « Prepare » créera deux lots d'images. Le premier lot créé servira à l'application d'apprendre à identifier les tumeurs. Le deuxième lot sera utile pour tester notre application en lui envoyant une image pour qu'il puisse déterminer si celle-ci contient une tumeur et à quel stade est cette tumeur.

Ensuite, après avoir réussi à générer les deux lots d'images, l'usager peut peser sur le boutton « Learn » qui aura pour but à l'application d'apprendre à déterminer quelle image est à quelle stade. Il faudra d'abord faire le traitement d'image. Pour ce faire, j'utilise le module *OpenCv* de *Python*. La première chose dont on doit faire est réduire le *noise* en rendant l'image floue. La raison pour laquelle je fais cela est qu'une image semble plus nette ou plus détaillée si nous sommes capables de percevoir correctement tous les objets et leurs formes. Par exemple, une image avec un visage semble plus claire lorsque nous sommes en mesure d'identifier très clairement les yeux, les oreilles, et etc. Cette forme d'un objet est due à ses bords. Donc, en rendant l'image plus floue nous réduisons simplement le contenu des bords et rendre la transition d'une couleur à l'autre très lisse. Puis, après avoir rendue l'image plus floue, nous utiliserons encore des fonctions de *OpenCv*, tels qu'une fonction qui détectera les *edges* (le nom de la fonction s'appelle *Canny()* dans *OpenCv*) et une autre qui détectera le contour des *edges* (la fonction s'appelle *findContours* dans *OpenCv*). La détection des « edges » nous aide à réduire la quantité de données (pixels) à traiter et maintient l'aspect structurel de l'image. *Canny()* trouve les limites ou les bords des objets dans une image, en déterminant où la luminosité de l'image change de façon spectaculaire. Cette étape doit être fait avant de pouvoir trouver les contours des objets. Un contour est une courbe joignant tous les points continus ayant la même couleur ou intensité. L'objectif est de recevoir un contour qui ressemble à un peu près à une ellipse. Par la suite, on devrait être en mesure de trouver l'aire de l'objet contourné. Puis, avec cela, on détermine les paramètres manquants pour définir une tumeur.

Après avoir fait le traitement d'images sur les images du premier lot, on utilisera un algorithme de *clustering* nommé *K-means* afin de regrouper les tumeurs trouvées par le traitement d'image et déterminer à quel stade de cancer ils se situent. La raison pour laquelle j'utilise l'algorithme *K-means* est que celle-ci est plus rapide comparé à l'autre algorithme de *clustering* nommé *Hierarchical Clustering*. Comme j'ai déjà déterminé le nombre de stades qu'il va y avoir, je sais alors le nombre de clusters que je voudrais créer et si jamais je décide de changer le nombre de stades, la méthode *K-means* est flexible pour les changements.

Chaque cluster aura son propre centripod. Les centripods sont considérés comme le centre du cluster. Nos centripods auront les paramètres idéals pour un certain stade de cancer. Chaque tumeur sera alors comparée à tous les centripods et ira rejoindre le cluster du centripod qui lui ressemble le plus. Il y aura aussi un centripod qui contiendra des paramètres qui correspondent à aucun stade de cancer. Ainsi, les tumeurs non-cancéreuses iront vers ce cluster. L'application pourra alors différencier les tumeurs entre elles.

Par la suite, l'usager pourra sélectionner une image spécifique du deuxième lot qui a été généré par l'application. En cliquant le bouton « Select » un *browser file* apparait directement sur le dossier du deuxième lot et l'usager pourra choisir quelle image il veut. Le chemin vers l'image apparaitra sur l'application.

Ensuite, l'application poursuivra le même processus pour le traitement d'image, mais avec seulement l'image spécifiée. Arrivé à ce stade, l'application a été entrainé à différencier les tumeurs. Donc, arrivé à ce stade, l'application pourra déterminer le stade de la tumeur en question.

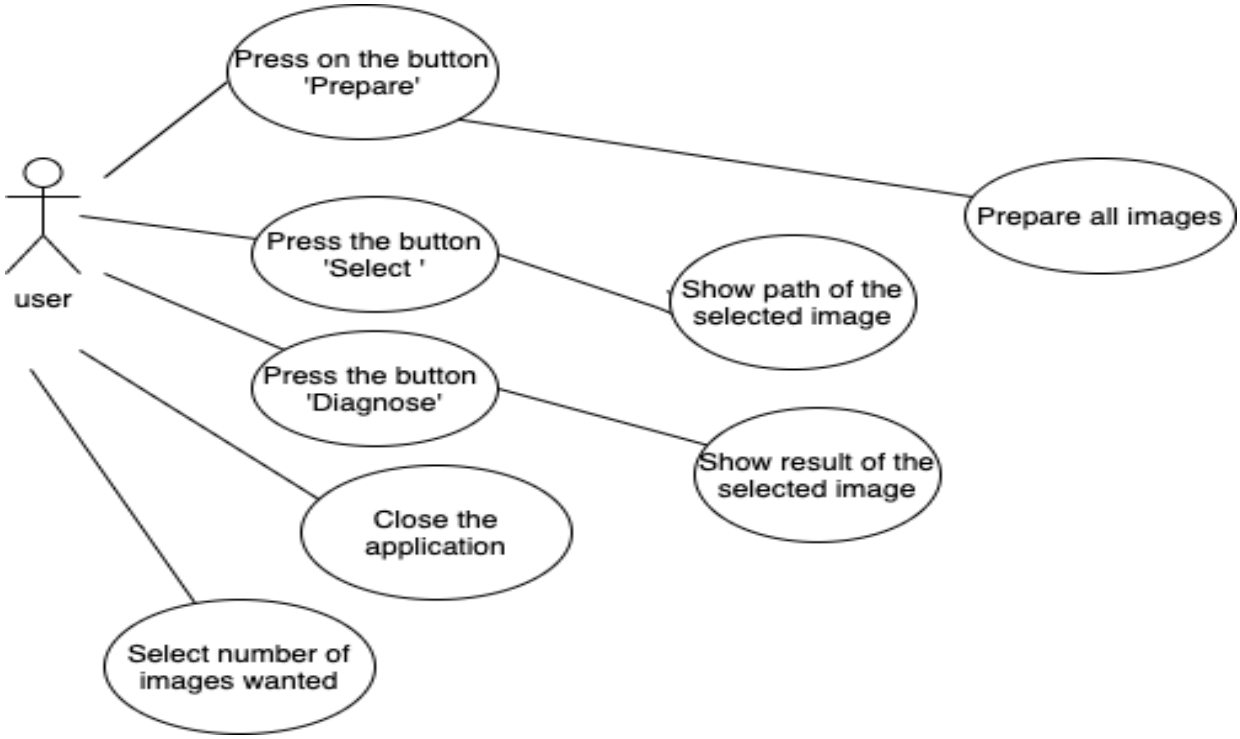
Présentation des patrons de conception

Pour ce projet, les deux patrons de conceptions que je vais utiliser sont **MVC**, et *Observer*. Tout d'abord, j'utilise **MVC**, car mon projet est un logiciel qui effectue des calculs et montre le résultat au moyen d'une interface graphique. Il est mieux de diviser la vue du modèle, car pour débugger cela sera plus facile de pouvoir savoir d'où vient le problème. Ce genre de projet peut évoluer et être modifié durant la session. Donc, avec **MVC**, modifier l'application au niveau du modèle ou de la vue devient pratique, car si je change au niveau de la vue, par exemple, je n'ai pas besoin d'aller modifier quoi que ce soit dans le modèle. Aussi, depuis que j'ai utilisé **MVC** personnellement, je trouve que séparer le modèle et la vue rendra le code plus lisible et plus flexible aussi. C'est la première fois que je fais un projet de A à Z et sur des concepts que je n'ai pas encore essayer au niveau de la vue et au niveau du modèle. Afin de ne pas me mélanger et de rendre le code illisible, j'ai décidé d'utiliser **MVC** afin de séparer en différents modules.

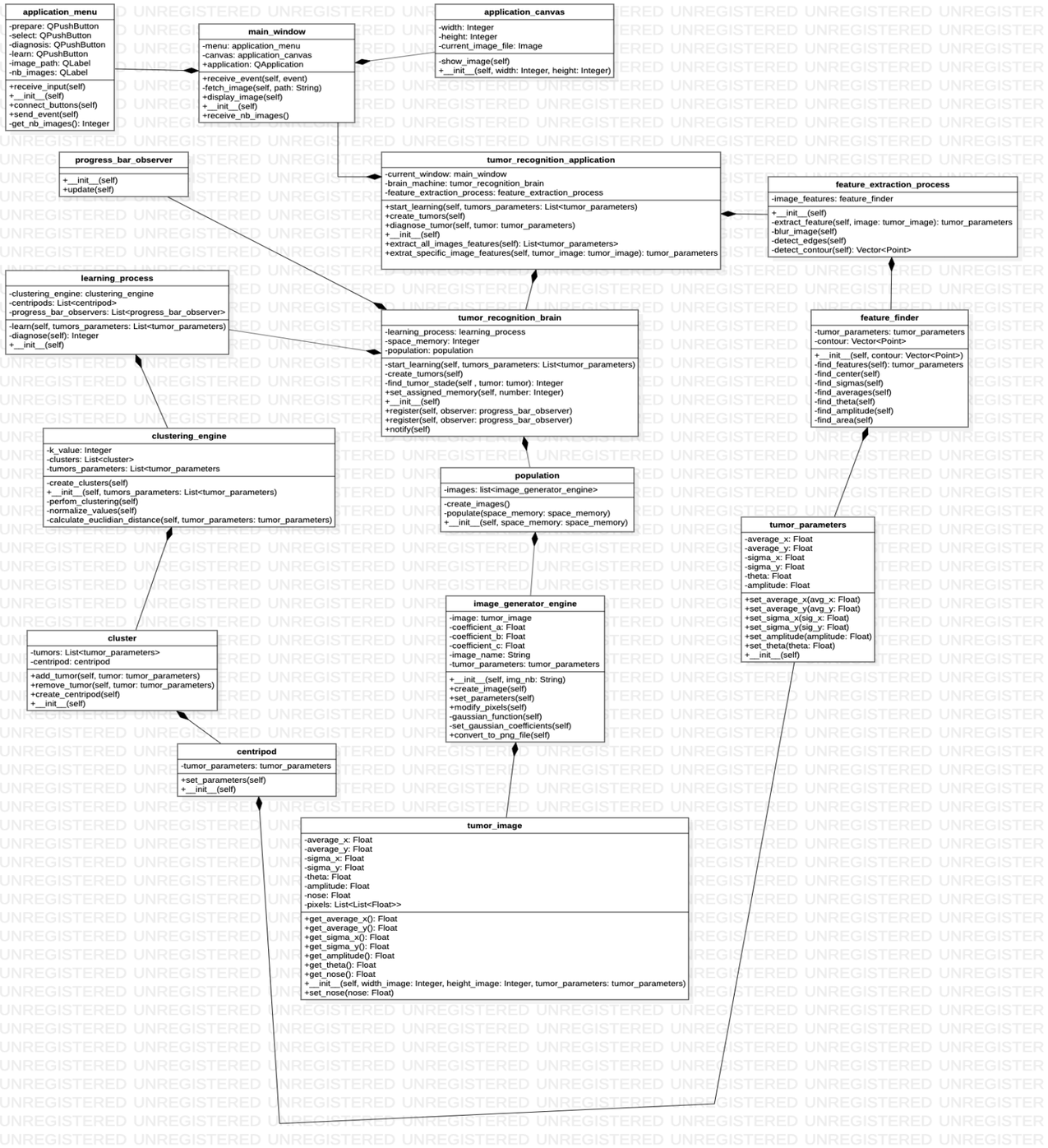
Mon deuxième patron de conception est *Observer.* J'ai choisi ce patron de conception, car durant tous les processus de mon application, je voudrais laisser savoir à l'usager où l'application est rendue grâce à la barre de progression. Pour l'instant, quand je génère mes deux lots d'images, il y a une barre de progression qui augmente de plus en plus jusqu'à ce que le processus de génération d'images soit terminé. Comme ça, l'usager ne vas pas commencer à cliquer sur d'autres boutons, si les images ne sont pas créées. Donc, mon *Observer* sera notifié quand il y a un changement dans mon l'application au niveau du processus d'images. Dépendamment d'où je suis rendue dans le projet durant la session, je voudrais aussi pouvoir observer où est rendue mon application dans le processus d'identifier et diagnostiquer la tumeur choisie par l'usager. Il y a d'autres possibilités intéressantes à envisager avec le patron de conception *Observer* et c'est pour cela que je l'ai choisi.

Aspects techniques de la conception

Cas d'utilisation



Diagramme(s) de conception technique



Conception des interfaces usagers