Christal

Compte rendu n°3

PSNR

Il est à noter que nous avons ajouté une fonction calculant le PSNR afin d'avoir une évaluation de nos résultats.

Interface graphique

Cette semaine, nous avons réfléchi à l'intégration de nos algorithmes dans une interface graphique.

Nous souhaitons ainsi pouvoir combiner nos filtres précédents créés, les algorithmes trouvés sur les papiers évoqués ainsi qu'une méthode qui requiert des réseaux de neurones. Toutes ces fonctionnalités seraient alors ajoutées au fur et à mesure de notre progression.

Pour le moment, nous voulons offrir le choix à l'utilisateur de modifier manuellement sa photographie par le biais de nos filtres et de manière interactive.

Nous allons utiliser QtCreator Py pour développer notre application. Christina ayant déjà utilisé la version C++ pour le projet image du master 1 et la version Py pour un autre projet totalement différent, c'est elle qui va tenter de mettre en place notre interface. Quelques soucis de compatibilité ont mis du temps à être réglés. En effet, QtCreator C++ fonctionne très bien sur la version ubuntu et QtCreator Py sur la version windows. Cependant, en voulant développer un nouveau projet sur ubuntu (car le code avait été créé sur ce système d'exploitation), et pour une raison inconnue, cela ne fonctionnait pas. Il a fallu passer sur windows pour que les modifications liées à l'interface graphique fonctionnent.

N'ayant jamais utilisé la version QtPy pour charger, afficher et modifier des images, nous allons d'abord nous pencher sur ces fonctionnalités. Il est à noter que les deux versions de QtCreator sont totalement différentes et nécessitent beaucoup de recherche pour aboutir à quelque chose de concret.

Pour charger notre image sur l'interface, nous avons d'abord créé un bouton *"bouton_ouvrirImgIn"* que nous avons connecté à la fonction ouvrirImage. Cette fonction que nous avons implémentée permet d'ouvrir un explorateur de fichiers (grâce à QFileDialog), qui va filtrer et ne montrer que les fichiers dont l'extension nous convient (ex: *.jpg), on sélectionne et valide l'image souhaitée. Ensuite, on récupère le chemin de cette image, on la convertit en QPixmap, et enfin, on l'affiche dans un label *"label_ImgIn"*.

La mise en place de ces éléments a été un peu laborieuse. Afin de clarifier la disposition des fonctions et dans le but de ne pas se retrouver avec un mainwindow.py de milliers de lignes, nous avons décidé qu'il serait plus judicieux de séparer certaines fonctions dans d'autres fichiers python. Cependant, malgré les "import .. from .." certains bugs généraient des conflits pas forcément évidents à comprendre et à corriger.

Actuellement nous avons 4 fichiers importants:

- form.ui : qui permet de modifier l'apparence de notre application
- mainwindow.py : qui est le main qui permet de gérer toutes les interactions Qt
- imageSettings.py : constituée des bases de la configuration d'images PIL
- imageGenNoises.py : regroupant les fonctions qui génèrent du bruit (celles-ci se basent sur nos codes précédents mais avec des modifications pour pouvoir être interactives avec Qt)

Afin d'illustrer nos propos, voici par exemple le code de la définition du bruit chromatique :

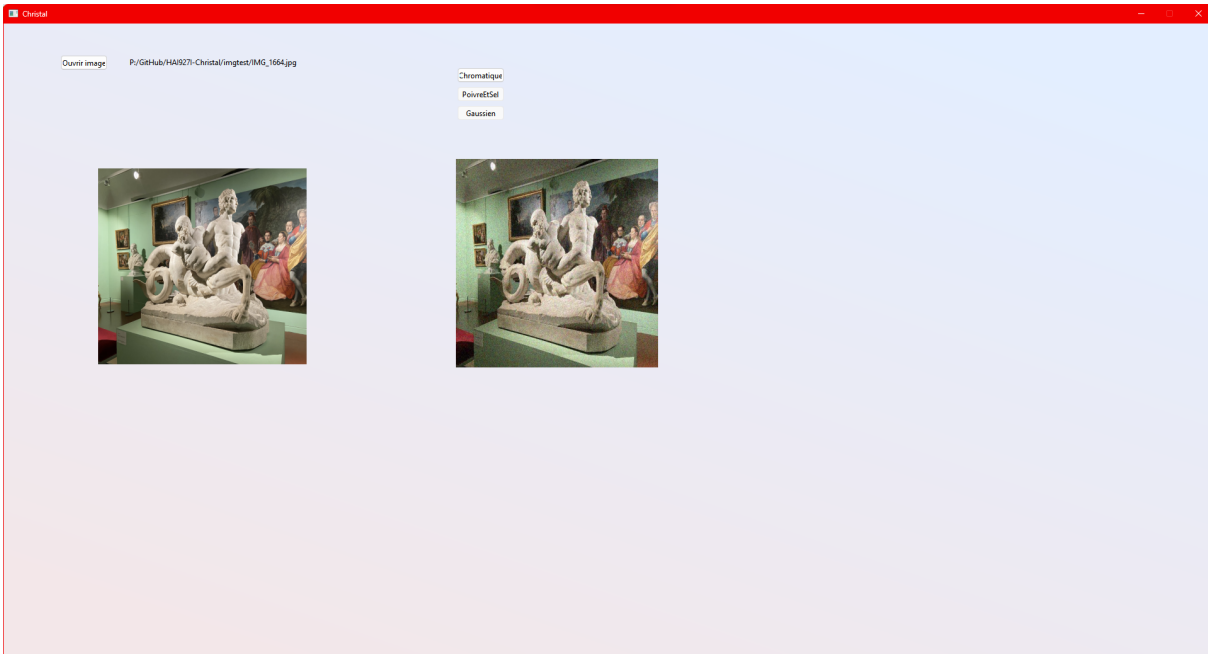
```
# Bruit chromatique
def bruit_chromatique(image, ecart_type, MainWindow):
    largeur, hauteur = image.size

    for x in range(largeur):
        for y in range(hauteur):
            bruit_r = int(np.random.normal(0, ecart_type))
            bruit_g = int(np.random.normal(0, ecart_type))
            bruit_b = int(np.random.normal(0, ecart_type))
            pixel = image.getpixel((x,y))
            r, g, b = pixel
```

```
            new_r = max(0, min(r + bruit_r, 255))
            new_g = max(0, min(g + bruit_g, 255))
            new_b = max(0, min(b + bruit_b, 255))
            new_pixel = (new_r, new_g, new_b)

            image.putpixel((x, y), new_pixel)
    enregistrerImageTmp(image)
    MainWindow.ImageModified = True
    print(MainWindow.ImageModified)
    MainWindow.affichageImageOut()
```

Nous avons décidé de créer une application qui prendrait tout l'espace écran (1920*1080) avec des couleurs très claires.



Ci-dessus un aperçu des fonctionnalités mises en place :

- Ouverture de l'image
- Affichage Qt de l'image
- Chargement de l'image en version PIL
- Appel des fonctions
- Sauvegarde de l'image modifiée dans le dossier temp
- Affichage de l'image modifiée

Il est à noter que les images ne sont pas encore mises à l'échelle correctement pour le moment, c'est surtout pour vérifier le fonctionnement de leurs labels.

Ci-dessous la définition de la fonction permettant d'ouvrir une image :

```
def ouvrirImage(self):
        options = QFileDialog.Options()
        fileName, _ = QFileDialog.getOpenFileName(self, "Sélectionner une image",
"", "Images (*.jpg *.jpeg)", options=options)
        if fileName:
            #Affichage du chemin de l'image sur l'UI
            self.ui.labeltext_chemin.setText(fileName)
            #Conversion Image en QPixmap
            pixmap = QPixmap(fileName)
            #Affichage de l'image dans un label
            self.ui.label_ImgIn.setPixmap(pixmap)
            self.ui.label_ImgIn.setScaledContents(True)

            #Définition de l'image en PIL
            image = ouvrirImageIn(fileName)

            self.ImageInIsSet = True
            self.ImageNdg = IsNdg(image)

            #Si l'image d'origine est définie
            if(self.ImageInIsSet):
                if(self.ImageNdg):
                    self.ui.bouton_poivresel.setEnabled(True)
                    self.ui.bouton_gaussien.setEnabled(True)
                else:
                    self.ui.bouton_chromatique.setVisible(True)
                    self.ui.bouton_chromatique.setEnabled(True)

            #Lien entre les boutons liés à "image" et les fonctions
            self.ui.bouton_poivresel.clicked.connect(lambda :
bruit_poivre_et_sel(image, self.densite, self))
            self.ui.bouton_gaussien.clicked.connect(lambda :
bruit_gaussien(image, self.ecart_type, self))
            self.ui.bouton_chromatique.clicked.connect(lambda :
bruit_chromatique(image, self.ecart_type, self))
```

Maintenant que nous avons réussi à mettre la base de notre code sous QtCreator en place, il sera plus aisé d'y ajouter la suite.

Efficient Poisson Denoising for Photography

Pendant cette semaine, nous avons également étudié plus en profondeur le papier *Efficient Poisson Denoising for Photography* .

Ce papier traite du bruit que l'on peut avoir dans les capteurs d'images, il met en évidence que ce bruit est dominé par des statistiques de Poisson. Le problème, c'est que de nombreuses méthodes de débruitages supposent souvent que le bruit présent est un bruit gaussien additif. Le bruit de Poisson peut dans certains cas être simplifié en utilisant une méthode pour stabiliser la variance, dans ce papier, ils vont utiliser la transformation d'Anscombe. Le problème, c'est qu'il faut avoir le nombre de photons pour pouvoir l'utiliser. Les auteurs proposent donc dans ce papier d'estimer le nombre de photons grâce aux caractéristiques de la distribution de Poisson afin de pouvoir effectuer une méthode pour stabiliser la variance et permettre au bruit de se rapprocher d'un bruit Gaussien, ce qui va permettre d'améliorer l'efficacité des méthodes de filtrage.

Ce papier présente quelques sources de bruit liées aux capteurs d'image, mais il se concentre principalement sur le bruit de photon. Ce bruit est causé par la nature discrète des photons eux-mêmes. Ils arrivent de manière aléatoire sur les capteurs, ce qui crée une variabilité dans le nombre de photons enregistrés à un pixel donné. Les auteurs ont décidé de se concentrer là-dessus, car c'est le bruit qui a le plus d'influence sur l'image de sortie.

Le bruit de Poisson est plus complexe qu'un bruit Gaussien, les méthodes de traitement d'images basées sur le bruit gaussien sont donc moins adaptés à ce type de bruit donc là que la transformation Anscombe est intéressante. Cette méthode va permettre de stabiliser la variance d'une distribution de Poisson, ce qui va permettre de rendre le bruit de Poisson plus semblable à un bruit gaussien additif. Voilà comment fonctionne la transformation Anscombe, nous avons ici P une variable aléatoire suivant une distribution de Poisson, la transformation est donc définie comme suit$A : P \rightarrow 2\sqrt{P + \frac{3}{8}}$.

Il va donc être nécessaire d'estimer le nombre de photons pour pouvoir appliquer notre transformation. Les photons sont transformés en pixel à travers plusieurs processus, l'un d'eux est l'AGC (Automatic gain control). Cette AGC va ajuster l'intensité moyenne du signal de sortie en fonction des conditions d'éclairage. Si on disposait du facteur de gain et de la résolution du capteur, on pourrait déterminer le nombre de photons, mais ces informations ne sont généralement pas disponibles même en utilisant un format RAW. La relation entre l'image I avant AGC et l'image J après AGC est la suivante $J = \gamma * I$ où γ est le

facteur de gain. En considérant que la distribution de Poisson du bruit est préservée après AGC, on peut assumer la relation suivante $E[T[I]] = Var[T[I]]$ où T(I) représente les zones sans textures complexes. On se retrouve avec $\gamma = \frac{E[T[J]]}{Var[T[J]]}$ et grâce à ça on va pouvoir déterminer le nombre de photons.

Il va tout de même valoir, déterminer les zones sans textures d'images, car il est difficile d'estimer la variation de bruit dans des zones texturées complexes. Dans ce papier ils proposent d'utiliser la formule suivante pour déterminer ces zones $T - [J] = \varepsilon w[\nabla[G\sigma * J] > \theta]$ où $G\sigma$ est une convolution gaussienne de variance σ, εw est une érosion morphologique et θ un paramètre d'intensité.

Et on peut finalement déterminer le nombre de photons avec la formule suivante $\Phi[J] = \frac{\sum(x,y)\epsilon I\, J(x,y)}{\gamma}$.

Ils ont ensuite fait des tests et leurs méthodes pour estimer les photons est plutôt efficace. Il suffit ensuite d'appliquer un algorithme de filtrage, la plupart des filtres qui assument que le bruit de l'image est un bruit Gaussien devrait bénéficier de cette transformation. Dans ce papier, ils utilisent un filtre bilatéral pour tester leur méthode.

Pour implémenter cette méthode, il va donc falloir faire ça en plusieurs étapes, tout d'abord mettre en place la méthode pour récupérer les zones sans textures, ensuite calculer notre facteur de gain puis notre nombre de photons, enfin, il faudra utiliser la transformation Anscombe et tester avec nos différents algorithmes de filtrages.

Cette méthode sera intéressante, nous pourrons comparer les résultats avec et sans cette transformation pour différentes méthodes afin de voir si les résultats sont améliorés.

Pour la semaine prochaine

Nous continuerons à développer notre application, intégrer les fonctionnalités déjà implémentées en python et améliorer son apparence afin que

ce soit lisible pour l'utilisateur. Nous essayerons également d'implémenter la transformation du papier que nous avons étudié.

Sources

-Efficient poisson denoising for photography; https://www.researchgate.net/profile/Hugues-Talbot-2/publication/224115039_Efficient_Poisson_Denoising_for_Photography/links/0deec525c4e064a606000000/Efficient-Poisson-Denoising-for-Photography.pdf